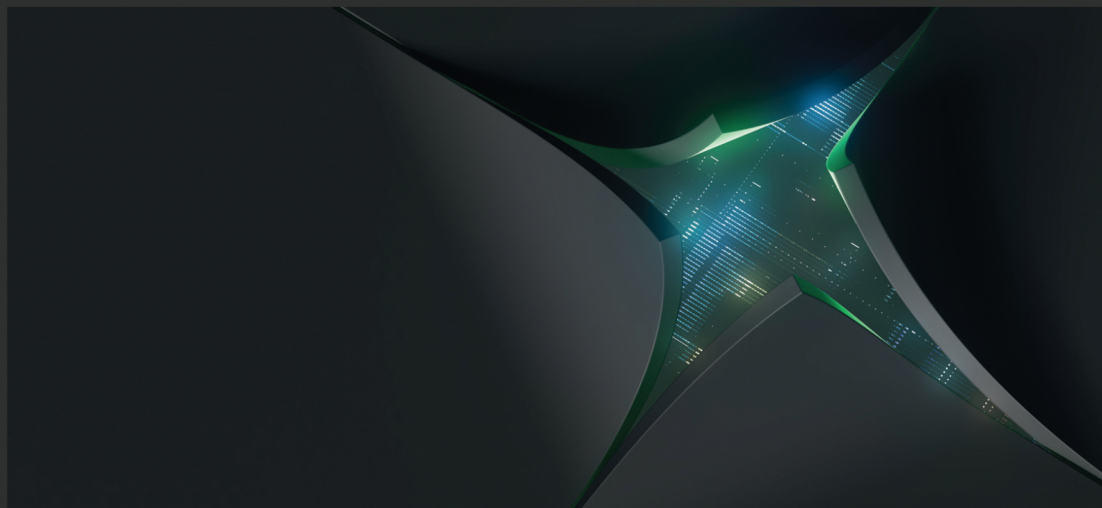


Второе издание

«packt»



Linux

для администраторов

Прокачайте навыки системного администрирования,
настраивая и обслуживая системы Linux



SPRINT
book

АЛЕКСАНДРУ КАЛКАТИНЖЕ
ДЖУЛИАН БАЛОГ

Mastering Linux Administration

Second Edition

Take your sysadmin skills
to the next level by configuring
and maintaining Linux systems

Alexandru Calcatine
Julian Balog

<packt>

Linux для администраторов

Второе издание

Прокачайте навыки системного
администрирования, настраивая
и обслуживая системы Linux

Александру Калкатинже
Джулиан Балог

Александр Калкатиңже, Джулиан Балог

Linux для администраторов

2-е издание

Перевел с английского Р. Чебыкин

Научный редактор Д. Квист

ББК 32.973.2-018.2

УДК 004.451

Калкатиңже Александр, Балог Джулиан

K17 Linux для администраторов, 2-е изд. — Астана: «Спринт Бук», 2026. — 672 с.: ил.

ISBN 978-601-12-8020-4

Раскройте мощь операционной системы Linux, используя ее непревзойденную универсальность для эффективного управления рабочими нагрузками как в локальных, так и в облачных средах. В книге представлены последние достижения в администрировании Linux. Устанавливайте Linux на локальные системы, освоите командную строку Linux, работу с файлами, пакетами и файловыми системами. Изучите ключевые команды и методы обеспечения безопасности вашей Linux-среды. В новом издании появилась глава по написанию сценариев командной оболочки, содержащая структурированные рекомендации для базовой автоматизации задач в Linux.

Уверенно выполняйте повседневные административные задачи и легко управляйте рабочими процессами — от локальной инфраструктуры до облачных сред.

16+ (В соответствии с Федеральным законом от 29 декабря 2010 г. № 436-ФЗ.)

© Packt Publishing 2024.

First published in the English language under the title 'Mastering Linux Administration - Second Edition – (9781837630691)'

© Перевод на русский язык «Спринт Бук», 2026

© Издание на русском языке, оформление «Спринт Бук», 2026

ISBN 978-601-12-8020-4

Изготовитель: ТОО «Спринт Бук». Место нахождения и фактический адрес:

010000, Казахстан, город Астана, район Алматы, проспект Рахымжан Кошкарбаев, д. 10/1, н. п. 18.

Дата изготовления: 05.2026. Наименование: книжная продукция. Срок годности: не ограничен.

Подписано в печать 27.03.26. Формат 70×100/16. Бумага офсетная. Усл. п. л. 54,180. Тираж 1000. Заказ 0000.

Краткое содержание

Часть 1

Основы администрирования Linux

https://t.me/it_books/2

Глава 1. Как установить Linux	24
Глава 2. Командная оболочка и файловая система Linux	55
Глава 3. Как управлять программным обеспечением в Linux.....	99
Глава 4. Как управлять пользователями и группами.....	135
Глава 5. Как работать с процессами, демонами и сигналами	176

Часть 2

Расширенные возможности администрирования Linux

Глава 6. Как работать с дисками и файловыми системами.....	206
Глава 7. Сетевое окружение Linux.....	234
Глава 8. Сценарии командной оболочки в Linux.....	282
Глава 9. Как обеспечивать безопасность в Linux.....	339
Глава 10. Аварийное восстановление, диагностика и устранение неполадок	392

Часть 3

Серверное администрирование

Глава 11. Как работать с виртуальными машинами.....	424
Глава 12. Как управлять контейнерами с помощью Docker	449
Глава 13. Как настраивать серверы Linux.....	470

Часть 4

Облачное администрирование

Глава 14. Краткое введение в облачные вычисления.....	494
Глава 15. Как разворачивать Linux в облаке с помощью AWS и Azure.....	515
Глава 16. Как разворачивать приложения с помощью Kubernetes	560
Глава 17. Как управлять инфраструктурой и автоматизацией с помощью Ansible	609

Оглавление

От издательства.....	16
О научном редакторе русского издания.....	16
Создатели книги	17
Об авторах	17
О рецензентах.....	18
Предисловие	19
Для кого эта книга	19
О чем эта книга	19
Как извлечь максимальную пользу из этой книги	21
Как загрузить файлы с кодом примеров	21
Условные обозначения	22

Часть 1 Основы администрирования Linux

Глава 1. Как установить Linux	24
Технические требования.....	24
Linux и его дистрибутивы	25
Дистрибутивы Linux.....	25
Как выбрать дистрибутив Linux	28
Устанавливаем Linux: первые шаги	30
Как установить Linux на физическое оборудование	31
Как установить Linux в виртуальной машине	34
Как оснастить виртуальную машину с помощью Hyper-V	40
Как оснастить виртуальную машину с помощью Oracle VirtualBox	42
Как работать с подсистемой Windows для Linux (WSL)	44
Устанавливаем Linux: продвинутые приемы.....	47
Процедура загрузки Linux	47
Как работает сетевая загрузка с помощью PXE	48
Дистрибутивы Linux: практическое руководство.....	50
Пример из практики: рабочая станция программиста	50
Пример из практики: защищенный веб-сервер	51
Пример из практики: личный блог	52
Пример из практики: медиасервер.....	52
Итоги	53
Вопросы	54
Дополнительная литература	54

Глава 2. Командная оболочка и файловая система Linux	55
Технические требования.....	55
Первое знакомство с оболочкой Linux	55
Как подключиться к командной оболочке	58
Приглашение командной строки.....	60
Типы команд оболочки.....	61
Как устроены команды Linux.....	62
Справочные страницы Linux.....	63
Файловая система Linux.....	64
Структура каталогов.....	65
Как работать с файлами и каталогами.....	68
Пути к файлам.....	68
Базовые операции с файлами	70
Как просматривать файлы	79
Как работать со свойствами файлов.....	82
Как создавать и редактировать файлы с помощью текстовых редакторов.....	94
Как редактировать текстовые файлы с помощью Vim.....	94
Текстовый редактор nano.....	96
Итоги	97
Вопросы	97
Дополнительная литература	98
Глава 3. Как управлять программным обеспечением в Linux	99
Технические требования.....	99
Типы пакетов программного обеспечения в Linux.....	99
Типы пакетов deb и RPM	100
Типы пакетов Snap и Flatpak.....	104
Как управлять пакетами программного обеспечения	107
Как управлять пакетами deb	107
Как управлять пакетами RPM.....	114
Как работать с пакетами Snap и Flatpak.....	123
Как устанавливать среды рабочего стола в Linux.....	129
Как установить KDE Plasma на Fedora Linux	130
Итоги	133
Вопросы	134
Дополнительная литература	134
Глава 4. Как управлять пользователями и группами	135
Технические требования.....	135
Как управлять пользователями	136
Что такое sudo	136
Как создавать, изменять и удалять учетные записи пользователей.....	138

Как управлять группами.....	149
Как создавать, изменять и удалять группы.....	149
Как управлять правами доступа	162
Права доступа к файлам и каталогам.....	163
Итоги	174
Вопросы	175
Дополнительная литература	175
Глава 5. Как работать с процессами, демонами и сигналами.....	176
Технические требования.....	177
Что такое процессы	177
Типы процессов в Linux	177
Как устроены процессы.....	183
Как работать с процессами	188
Как просматривать сведения о процессах с помощью ps	189
Как просматривать дерево процессов с помощью команды pstree	191
Как просматривать сводку процессов с помощью команды top	192
Как принудительно завершать процессы с помощью команд kill и killall	194
Использование команд pgrep и pkill.....	195
Как работать с демонами	196
Как работать с демонами systemd.....	197
Межпроцессное взаимодействие.....	198
Сигналы в Linux	201
Итоги	203
Вопросы	203
Дополнительная литература	204

Часть 2

Расширенные возможности администрирования Linux

Глава 6. Как работать с дисками и файловыми системами.....	206
Технические требования.....	206
Устройства в Linux.....	206
Уровни абстракции Linux.....	207
Файлы устройств и соглашения об именах	208
Файловые системы в Linux.....	211
Диски и разделы	215
Распространенные типы дисков	215
Как разбивать диски на разделы.....	217
Как управлять логическими томами в Linux	228
Снимки логических томов	231
Итоги	232
Вопросы	233

Глава 7. Сетевое окружение Linux	234
Технические требования.....	234
Основы сетевых технологий	234
Компьютерные сети.....	235
Модель OSI	236
Модель сетевого стека TCP/IP	241
Протоколы TCP/IP	243
IP-адреса.....	245
Сокеты и порты.....	249
Как настраивать сетевое окружение в Linux	251
Как работать с сетевыми службами в Linux	260
Серверы DHCP	261
Серверы DNS	262
Серверы аутентификации	265
Совместный доступ к файлам.....	266
Серверы печати.....	268
Передача файлов	268
Почтовые серверы.....	269
Серверы NTP	272
Удаленный доступ.....	274
Задачи сетевой безопасности.....	279
Итоги	280
Вопросы	280
Дополнительная литература	281
Глава 8. Сценарии командной оболочки в Linux	282
Технические требования.....	282
Командная оболочка Linux.....	283
Возможности командной оболочки Bash.....	283
Переменные оболочки Bash.....	288
Введение в сценарии оболочки	290
Как создать файл сценария оболочки.....	291
Переменные в сценариях оболочки	293
Как использовать математические выражения в сценариях оболочки.....	295
Управляющие конструкции в Bash	297
Массивы в Bash.....	297
Как считывать входные данные	299
Как форматировать выходные данные	300
Коды завершения и проверка условий	303
Условные выражения if	305
Циклы Bash	307
Как работать с функциями	315
Команды sed и (g)awk.....	319

Демонстрация межпроцессного взаимодействия с помощью сценариев.....	322
Общее хранилище	322
Неименованные каналы.....	324
Именованные каналы	325
Сокеты.....	327
Как выполнять административные задачи с помощью сценариев.....	329
Как создавать сценарии для задач системного администрирования	329
Как создавать пакеты для сценариев.....	333
Итоги	337
Вопросы.....	338
Дополнительная литература	338
Глава 9. Как обеспечивать безопасность в Linux	339
Технические требования.....	340
Как устроена безопасность в Linux.....	340
Система управления доступом SELinux.....	341
Как устроен SELinux.....	341
Модуль безопасности AppArmor	360
Как работать с AppArmor.....	360
Заключительные соображения.....	366
Межсетевые экраны в Linux.....	367
Цепочки в межсетевых экранах	368
Представляем Netfilter	370
Как работать с iptables.....	372
Как работать с nftables.....	377
Системы управления межсетевыми экранами.....	381
Итоги	390
Упражнения.....	391
Дополнительная литература	391
Глава 10. Аварийное восстановление, диагностика и устранение неполадок.....	392
Технические требования.....	392
Как планировать аварийное восстановление	392
Краткое введение в управление рисками	393
Расчет рисков.....	395
Как разрабатывать план аварийного восстановления.....	397
Резервное копирование и восстановление системы	398
Решения для клонирования дисков	399
Популярные инструменты для диагностики и устранения неполадок в Linux	404
Как устранять неполадки загрузки.....	405
Как устранять неполадки, связанные с потреблением системных ресурсов	407

Как устранять неполадки, связанные с сетью	414
Как устранять неполадки оборудования	420
Итоги	422
Вопросы	422

Часть 3

Серверное администрирование

Глава 11. Как работать с виртуальными машинами	424
Технические требования	424
Введение в виртуализацию в Linux	425
Как виртуализация помогает эффективнее использовать ресурсы	425
Что такое гипервизоры	427
Как устроен гипервизор KVM в Linux	427
Особенности KVM	428
Как работать с гипервизором KVM	428
Как работать с основными командами KVM	432
Как создать виртуальную машину с помощью командной строки	432
Как управлять виртуальными машинами	433
Расширенное управление KVM	434
Как подключиться к виртуальной машине	435
Как клонировать виртуальные машины	437
Как работать с шаблонами виртуальных машин	438
Как получать сведения о ресурсах виртуальных машин и узлов	439
Как управлять использованием ресурсов виртуальной машины	441
Как управлять виртуальными машинами с помощью cloud-init	442
Как работает cloud-init	442
Как установить и настроить cloud-init	442
Аутентификация SSH с открытым ключом	445
Итоги	447
Вопросы	447
Дополнительная литература	448
Глава 12. Как управлять контейнерами с помощью Docker	449
Технические требования	449
Что такое контейнеры Linux	449
Контейнеры и виртуальные машины	449
Основы контейнерных технологий	451
Что такое Docker	453
Основы работы с Docker	456
Какую версию Docker выбрать	457
Как установить Docker	457

Некоторые команды Docker	459
Как управлять контейнерами Docker.....	461
Как работать с Docker-файлами	464
Как создать контейнерный образ с помощью Docker-файла.....	465
Как развертывать контейнерные приложения с помощью Docker	467
Как развернуть сайт с помощью Docker	467
Итоги	469
Вопросы	469
Дополнительная литература	469
Глава 13. Как настраивать серверы Linux	470
Технические требования.....	470
Введение в службы Linux.....	471
Как настроить сервер SSH	473
Как установить и настроить OpenSSH в Ubuntu.....	473
Как настроить сервер DNS.....	474
Как настроить кэширующий сервер DNS.....	475
Как настроить первичный сервер DNS	477
Как настроить вторичный сервер DNS.....	479
Как настроить сервер DHCP	482
Как настроить сервер NFS	484
Как установить и наладить сервер NFS.....	484
Как настроить клиент NFS	486
Как протестировать конфигурацию NFS.....	487
Как настроить файловый сервер Samba.....	487
Как установить и наладить сервер Samba.....	488
Как создавать учетные записи Samba	490
Как получить доступ к общим ресурсам Samba	491
Итоги	492
Вопросы	492
Дополнительная литература	492

Часть 4

Облачное администрирование

Глава 14. Краткое введение в облачные вычисления.....	494
Технические требования.....	494
Введение в облачные технологии.....	494
Стандарты облачных вычислений	495
Архитектура облачных решений	499
Ключевые особенности облачных вычислений.....	501
IaaS: инфраструктура как услуга.....	502
Amazon EC2.....	503

Виртуальные машины Microsoft Azure.....	503
Альтернативные предложения IaaS.....	504
PaaS: платформа как услуга.....	505
Amazon Elastic Beanstalk.....	505
Google App Engine.....	505
App Platform от DigitalOcean.....	506
Решения PaaS с открытым исходным кодом.....	506
SaaS: контейнеризация как услуга.....	507
Введение в Kubernetes.....	507
Развертывание контейнеров в облачной среде.....	509
Микросервисы в облачных вычислениях.....	510
Что такое DevOps.....	511
Инструменты управления облачными средами.....	512
Ansible.....	512
Puppet.....	513
Chef Infra.....	513
Итоги.....	514
Дополнительная литература.....	514
Глава 15. Как развертывать Linux в облаке с помощью AWS и Azure	515
Технические требования.....	516
AWS EC2.....	517
Как создавать и оснащать экземпляры AWS EC2.....	517
Группы размещения AWS EC2.....	530
Экземпляры AWS EC2 на практике.....	531
Интерфейс командной строки AWS CLI.....	540
Microsoft Azure.....	546
Как создать и развернуть виртуальную машину.....	547
Как подключиться к виртуальной машине с помощью SSH.....	552
Как управлять виртуальными машинами.....	553
Интерфейс командной строки Azure CLI.....	555
Итоги.....	558
Вопросы.....	559
Дополнительная литература.....	559
Глава 16. Как развертывать приложения с помощью Kubernetes	560
Технические требования.....	560
Введение в Kubernetes.....	561
Архитектура Kubernetes.....	562
Объектная модель Kubernetes.....	563
Как устроен кластер Kubernetes.....	566
Как установить и настроить Kubernetes.....	569
Как установить Kubernetes на настольный компьютер.....	569

Как создать кластер Kubernetes из виртуальных машин	570
Как работать с Kubernetes.....	584
Утилита командной строки kubectl.....	584
Как развертывать приложения в Kubernetes.....	591
Как запускать Kubernetes в облаке	606
Итоги	606
Вопросы.....	607
Дополнительная литература.....	607
Глава 17. Как управлять инфраструктурой и автоматизацией	
с помощью Ansible.....	609
Технические требования.....	609
Архитектура и управление конфигурацией Ansible	610
Введение в архитектуру Ansible.....	610
Что такое управление конфигурацией	613
Как установить Ansible.....	614
Как установить Ansible в Ubuntu	615
Как установить Ansible с помощью pip.....	616
Как работать с Ansible	617
Учебная среда для Ansible	617
Как настраивать Ansible.....	619
Разовые команды Ansible	624
Обзор модулей Ansible	632
Как работать со сценариями Ansible	633
Как использовать шаблоны Jinja2 в сценариях Ansible.....	658
Роли Ansible	664
Итоги	670
Вопросы.....	671
Дополнительная литература.....	671

Посвящается всем, кто не верил в меня, а также всем, кто верил.

*Моим маме и папе, сестре-близняшке, моей любимой Юке
и моим верным друзьям.*

*В память о моем покойном дедушке Мироне, который мечтал дожить
до того дня, когда эта книга выйдет в свет.*

От издательства

Мы выражаем огромную благодарность компании Orion soft за помощь в работе над русскоязычным изданием книги и вклад в повышение качества переводной литературы. Редакторы русскоязычного издания приложили немало усилий, чтобы актуализировать и дополнить информацию, приведенную авторами книги, уточнить фрагменты кода и исправить ряд ошибок, допущенных в оригинальном издании.

Ваши замечания, предложения, вопросы отправляйте по адресу comp@sprintbook.kz (издательство «SprintBook», компьютерная редакция).

Мы будем рады узнать ваше мнение!

О научном редакторе русского издания

Денис Квист — эксперт по виртуализации в компании Orion soft с более чем 24-летним опытом работы в IT. Его профессиональный путь начался с должности системного администратора и программиста в Национальном архиве Республики Карелия, где он стоял у истоков цифровизации архивной отрасли. За свою карьеру занимал ведущие позиции в различных организациях, включая гостиницу ParkInn, Республиканский Перинатальный центр, Республиканский медицинский информационно-аналитический Центр. Под его руководством более двухсот студентов прошли обучение по основам Linux-систем на платформе Geek Brains. Сегодня он применяет знания как в личных проектах (обслуживает и развивает три FM-радиостанции), так и в профессиональной среде: активно участвует в разработке и поддержке системы виртуализации zVirt.

Создатели книги

Об авторах

Александр Калкати́нже — архитектор широкого профиля с образованием в области математики и computer science. Старший преподаватель университета, получил степень PhD в области городского планирования в Университете архитектуры и градостроительства имени Иона Минку и степень магистра по специальности DevOps в Учебном центре в области технологий и менеджмента (СТМЕ) при Калифорнийском технологическом институте. Александру обучает студентов архитектурному программированию и разработке, а также технологиям с открытым исходным кодом. Он написал пять книг по архитектуре и градостроительству и множество научных статей о развитии городов и сельских районов. В 2017 году получил сертификат преподавателя Linux. Александру вдохновлен философией DevOps и перспективными возможностями облачных технологий. Кроме того, он дипломированный программист-аналитик, сетевой администратор, преподаватель, проектировщик и консультант по персональному развитию.

Хочу поблагодарить людей, которые были рядом и безоговорочно поддерживали меня, — особенно родителей, сестру и друзей. Я особенно признателен моей единственной и неповторимой Юке за то, что она оставалась со мной в трудные времена, непреклонно верила в меня и поддерживала мои начинания.

Джулиан Балог — старший программист с опытом работы более 15 лет. Сейчас он в основном занимается контроллерами доставки приложений, контейнерными рабочими процессами, сетевыми технологиями и безопасностью. С неиссякаемой страстью к Linux и технологиям с открытым исходным кодом Джулиан всегда стремится узнать что-то новое, когда решает проблемы и внедряет простые, эффективные и практичные инженерные решения. Он живет с женой, двумя детьми и австралийским пуделем в пригороде Сизтла.

Авторы благодарны замечательной редакционно-издательской команде Packt за профессиональное руководство, преданность делу и наставления в ходе работы над этой книгой. Низкий поклон этим людям за то, что тщательно просматривали черновики и внесли множество полезных предложений и критических замечаний. О лучшей команде и поддержке мы и мечтать не могли!

О рецензентах

Химаншу Шарма (Himanshu Sharma) почти 18 лет занимается проектированием, архитектурой и разработкой облачного и сетевого программного обеспечения. Ранее он работал в таких крупных компаниях, как Brocade и Juniper, а также в таких стартапах, как Netskope. Сейчас он занимает должность ведущего инженера в Netskope и отвечает за услуги в сфере безопасности. Химаншу с нуля спроектировал и разработал передовые службы Netskope для защиты от угроз. Его профессиональные интересы и опыт связаны с разработкой масштабируемых облачных сервисов с помощью передовых технологий. В свободное время он катается на лыжах и играет в видеоигры.

Я хотел бы поблагодарить свою жену Пуджу, которая оказывала мне всяческую поддержку, и двух наших любящих и прекрасных дочерей — Навью и Джаанви.

Также хочу поблагодарить своего брата Судханшу за то, что он всегда меня прикрывал, и своих родителей за все, чем они пожертвовали, чтобы я обрел свое место в жизни.

Деннис Саламанка (Dennis Salamanca) — страстный технический энтузиаст со стажем работы в сфере IT более 12 лет. На протяжении всей карьеры ему повезло работать в компаниях, лидирующих в отрасли, — таких как Amazon, VMware, Microsoft и Hewlett Packard Enterprise. Деннис постоянно стремится учиться новому, что отразилось в его коллекции из более чем 15 сертификатов в разных технических областях, включая облачные технологии, хранение данных, Linux, Kubernetes и виртуализацию. Помимо прочего, он активно участвует в разработке программ сертификации Linux+ и Cloud+ и входит в число экспертов в предметных областях Linux+ и Cloud+ ассоциации CompTIA, а также в состав Технического консультативного комитета CompTIA.

Сердечно благодарю свою жену и семью за поддержку на протяжении всех этих лет. Без мотивации и содействия с их стороны ничего бы не получилось.

Предисловие

«Linux для администраторов» — это всеобъемлющий обзор современных технологий администрирования серверов и облачных сред.

Технологии развиваются с небывалой скоростью, а Linux и все, что с ним связано, находится на переднем крае инноваций — поэтому нелегко успевать за изменениями и учиться новому. Современные администраторы Linux должны разбираться не только в самой операционной системе: будущим специалистам по DevOps не менее важно ориентироваться в контейнеризации и облачных технологиях.

Linux — это система, под управлением которой работает практически все: от «интернета вещей» до персональных компьютеров и серверов; кроме того, на Linux опираются все облачные технологии, что позволяет управлять ими из командной строки.

Сначала вы научитесь работать с командной строкой, файлами, процессами, пользователями, пакетами и файловыми системами, затем начнете администрировать сетевые службы и обеспечивать их безопасность и, наконец, освоите облачные вычисления, контейнеры и оркестровку. Вы узнаете о самых важных командах Linux, которые позволяют управлять пользователями, процессами и службами, а также сможете обеспечить безопасность Linux с помощью iptables. После этого вы поработаете с контейнерами, гипервизорами, виртуальными машинами, Ansible и Kubernetes и узнаете, как развернуть Linux на AWS и Azure. К концу книги вы вполне освоите Linux и сможете работать с этой системой в духе DevOps — от установки на «голом железе» до облачных экземпляров.

Для кого эта книга

Книга предназначена для системных администраторов Linux, которые хотят изучить основы и современные принципы администрирования этой системы. Книга будет полезна и тем системным администраторам Windows, кому интересно разобраться в Linux.

О чем эта книга

В главе 1 «Как установить Linux» рассказывается о том, как установить Linux на физическом оборудовании или на виртуальной машине в Windows. Поскольку книга рассчитана на будущих системных администраторов Linux, мы будем в основном работать в командной строке, практически не задействуя графический

интерфейс. Из этой главы будущие специалисты по Linux узнают, как установить Linux и как устроен процесс загрузки.

В главе 2 «Командная оболочка и файловая система Linux» вы научитесь пользоваться командной строкой и познакомитесь с наиболее распространенными командами Linux. Вы узнаете о том, как устроены простейшие команды, как организована файловая система Linux, какова структура операционной системы Linux и файлов в ней. К концу главы вы также научитесь работать с vi/Vim — одним из самых популярных текстовых редакторов Linux с интерфейсом командной строки.

Глава 3 «Как управлять программным обеспечением в Linux» посвящена тому, как управлять ПО с помощью важнейших команд, как устроены пакеты программ в разных дистрибутивах и как создавать собственные пакеты.

В главе 4 «Как управлять пользователями и группами» рассказывается о том, как управлять учетными записями пользователей в Linux. Это одно из самых важных умений, которыми должен владеть системный администратор. Вы освоите общие понятия, познакомитесь со специальными файлами для администрирования пользователей и узнаете, как управлять учетными записями. К концу главы вы научитесь работать с правами доступа и редактировать их, а также будете разбираться в специальных правах доступа и атрибутах.

Глава 5 «Как работать с процессами, демонами и сигналами» посвящена процессам, сигналам и службам в Linux. Вы узнаете, как управлять ими, как их использовать и чем они отличаются друг от друга.

Глава 6 «Как работать с дисками и файловыми системами» расскажет о том, как хранятся данные в Linux, научит вас управлять дисками и файловыми системами, использовать системы управления логическими томами, а также монтировать файловые системы и работать с разделами дисков.

Из главы 7 «Сетевое окружение Linux» вы узнаете, как устроена сеть в Linux, какие принципы лежат в основе сетевых служб и как настраивать сетевое окружение из командной строки и графического интерфейса.

Глава 8 «Сценарии командной оболочки в Linux» демонстрирует, как создавать и использовать сценарии оболочки Bash, чтобы автоматизировать задачи в Linux. Эти навыки станут неоценимым подспорьем для любого системного администратора.

Глава 9 «Как обеспечивать безопасность в Linux» посвящена продвинутым темам безопасности Linux. В частности, вы научитесь работать с SELinux и AppArmor.

В главе 10 «Аварийное восстановление, диагностика и устранение неполадок» вы узнаете, как выполнять резервное копирование системы и восстанавливать ее после аварийных сбоев, а также как диагностировать и устранять распространенные неполадки.

Глава 11 «Как работать с виртуальными машинами» рассказывает о том, как настраивать виртуальные машины KVM в Linux и работать с ними.

Из главы 12 «Как управлять контейнерами с помощью Docker» вы узнаете о том, что такое контейнеры и как развертывать приложения с помощью специфических инструментов Docker.

Глава 13 «Как настраивать серверы Linux» посвящена тому, как настраивать различные типы серверов Linux, начиная с серверов DNS, DHCP, SSH, Samba и NFS. Это один из основных навыков любого хорошего системного администратора.

Глава 14 «Краткое введение в облачные вычисления» охватывает основы облачных технологий. Вы познакомитесь с такими моделями обслуживания, как инфраструктура как услуга (IaaS), платформа как услуга (PaaS), контейнеры как услуга (SaaS), а также с принципами DevOps и инструментами управления облачной службой.

В главе 15 «Как развертывать Linux в облаке с помощью AWS и Azure» вы научитесь развертывать Linux в AWS и Azure.

Глава 16 «Как развертывать приложения с помощью Kubernetes» продемонстрирует, как с помощью Kubernetes отслеживать развертывания и обеспечивать их безопасность, а также управлять контейнерами и сетями. Вы узнаете, что такое Kubernetes и какие методы работы с ним сложились в профессиональном сообществе.

Глава 17 «Как управлять инфраструктурой и автоматизацией с помощью Ansible» рассказывает о системе управления конфигурациями Ansible — в том числе о том, как ее настраивать и как управлять сценариями, модулями и серверами. К концу этой главы вы станете мастером автоматизации.

Как извлечь максимальную пользу из этой книги

Чтобы выполнять примеры из этой книги, вам понадобится Ubuntu Linux LTS или Debian Linux. Никаких предварительных знаний о Linux не требуется.

Если вы используете цифровую версию книги, мы советуем вам набирать код самостоятельно или извлечь его из репозитория GitHub, ссылка на который указана в следующем разделе. Это поможет избежать потенциальных ошибок на этапе копирования и вставки кода.

Как загрузить файлы с кодом примеров

Файлы с кодом примеров из этой книги можно загрузить с GitHub по адресу github.com/PacktPublishing/Mastering-Linux-Administration-Second-Edition. В этом репозитории отражаются все обновления кода после выхода книги.

Обратите внимание, что издательство Packt предлагает и другие подборки кода из богатого каталога книг и видеороликов, доступного по адресу github.com/PacktPublishing/.

Условные обозначения

В этой книге используются условные обозначения.

Код в тексте. Обозначает фрагменты кода в тексте, имена таблиц в базах данных, имена файлов и папок, расширения файлов, пути в файловой системе, фиктивные URL-адреса, пользовательский ввод и псевдонимы Twitter, например: «Чтобы проверить содержимое пакета `deb`, можно использовать команду `ar`».

Блок кода, а также ввод и вывод в командной строке обозначаются так:

```
спес:
  replicas: 1
```

Когда мы хотим обратить ваше внимание на определенный фрагмент блока кода или команды, то соответствующие строки или элементы выделяются полужирным шрифтом:

```
uid=1004(alex2) gid=1100(admin)
groups=1100(admin),1200(developers),1300(devops),1400(managers)
```

Полужирный шрифт обозначает новый термин, важные слова или понятия.

Советы или важные примечания выглядят так.

Часть 1

ОСНОВЫ

АДМИНИСТРИРОВАНИЯ

LINUX

https://t.me/it_boooks/2

В первой части этой книги вы освоите командную строку Linux и основные административные задачи — в частности, как управлять пользователями, пакетами, файлами, службами, процессами, сигналами и дисками.

Эта часть состоит из следующих глав:

- *Глава 1. Как установить Linux*
- *Глава 2. Командная оболочка и файловая система Linux*
- *Глава 3. Как управлять программным обеспечением в Linux*
- *Глава 4. Как управлять пользователями и группами*
- *Глава 5. Как работать с процессами, демонами и сигналами*

1

Как установить Linux

В последние годы **Linux** становится все популярнее в качестве операционной системы как для серверов, так и для настольных компьютеров. Эта система стала стандартной платформой для широкого спектра приложений: от корпоративных серверов и крупномасштабных облачных инфраструктур до персональных рабочих станций и домашней бытовой техники.

Распространение Linux привело к тому, что в растущем сообществе системных администраторов и разработчиков навыки администрирования сейчас как никогда востребованы. В этой книге мы рассматриваем основы администрирования Linux с практической точки зрения, ориентируясь на современных системных администраторов, специалистов по DevOps и разработчиков.

Во втором издании мы немного изменили подход к установке Linux. Поскольку книга предназначена для продвинутых читателей, мы не будем разъяснять основы установки ОС так же подробно, как в первом издании. Мы обновили материал так, чтобы он отражал наиболее свежие версии операционных систем по состоянию на начало 2023 года.

В первой главе мы расскажем о том, как установить Linux на физическое оборудование или в **виртуальной машине**.

Мы рассмотрим такие темы:

- Linux и его дистрибутивы.
- Как установить Linux.
- Как включить подсистему Windows для Linux.
- Расширенные возможности установки Linux.
- Практическое руководство по дистрибутивам Linux.

Технические требования

В этой главе мы будем использовать такие платформы и технологии:

- Дистрибутив Linux: Ubuntu.
- Гипервизоры виртуальных машин: Oracle VirtualBox, VMware Workstation, Hyper-V.

- Управляющая система виртуальной машины: Windows 11 (также подойдет macOS).

Linux и его дистрибутивы

Linux — это относительно современная операционная система: ее создал в 1991 году Линус Торвалдс (Linus Torvalds), который тогда изучал computer science в Хельсинкском университете. Сперва Linux выпускался как свободная платформа с открытым исходным кодом и запретом на коммерческое распространение, однако в 1992 году он перешел на модель **GNU GPL**. Этот шаг во многом обусловил то, что Linux начал широко распространяться не только среди разработчиков, но и в корпоративной среде. Важно отметить, что в сообществе специалистов, связанных с Фондом свободного программного обеспечения (FSF), операционные системы (или дистрибутивы) Linux называют **GNU/Linux**, чтобы подчеркнуть, насколько важны инициативы GNU для свободного программного обеспечения.

Сначала Linux создавался для компьютерных архитектур на базе процессоров Intel x86, но впоследствии был перенесен на множество платформ и к настоящему времени стал одной из самых популярных операционных систем. Можно считать, что Linux возник как альтернатива с открытым исходным кодом своему могущественному предшественнику Unix — коммерческой операционной системе, которую разработали Кен Томпсон (Ken Thompson) и Деннис Ритчи (Dennis Ritchie) в Лабораториях Белла компании AT&T в 1969 году.

Дистрибутивы Linux

Операционные системы класса Linux обычно отождествляют с **дистрибутивами**. Дистрибутив Linux — это установочный пакет (как правило, образ ISO) операционной системы с набором инструментов, библиотек и дополнительных программных пакетов, которые устанавливаются поверх **ядра** Linux¹. Ядро служит основным интерфейсом, через который оборудование компьютера и его процессы взаимодействуют друг с другом и который эффективно управляет имеющимися ресурсами.

В набор программ, которые поставляются вместе с ядром Linux, обычно входит загрузчик, командная оболочка, системы управления пакетами, графический интерфейс пользователя, а также различные утилиты и приложения.

Вот упрощенная схема архитектуры типичного дистрибутива Linux:

¹ Ядро Linux (Linux kernel) разрабатывается и поддерживается сообществом разработчиков под руководством Линуса Торвалдса; официальные выпуски ядра размещаются на сайте kernel.org. Создатели разных дистрибутивов берут за основу разные версии ядра, по-своему их модифицируют и дополняют различными системными и прикладными компонентами. — *Примеч. ред.*

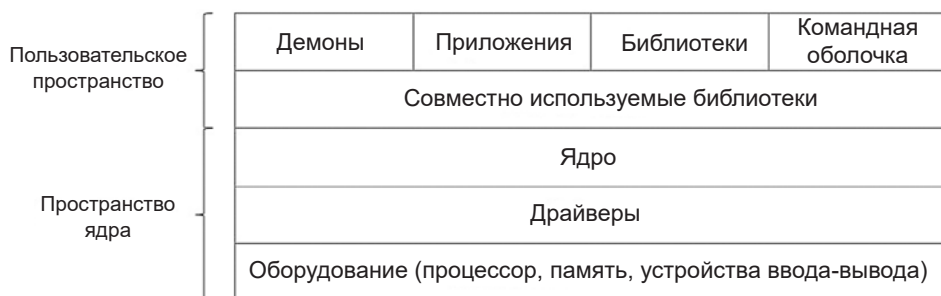


Рис. 1.1. Упрощенная схема типичной архитектуры Linux

В настоящее время существуют сотни дистрибутивов Linux. Среди самых старых и, пожалуй, самых популярных — **Debian**, **Fedora**, **openSUSE**, **Arch Linux** и **Slackware**, а также множество других дистрибутивов, которые основаны на перечисленных или являются производными от них. Одни дистрибутивы — коммерческие, а другие поддерживаются сообществом.

Когда готовилось второе издание этой книги, CentOS перешла на модель скользящих выпусков и стала основой, на которой базируются последующие версии **Red Hat Enterprise Linux (RHEL)**. Экологическую нишу CentOS заняли другие свободные дистрибутивы, которые поддерживаются сообществом и используют исходный код RHEL. Хороший пример такого дистрибутива — **Rocky Linux**, на который мы будем ссылаться в этой книге. Также на RHEL основан **AlmaLinux**, который тоже поддерживается сообществом.

Одно из ключевых различий между дистрибутивами Linux заключается в том, какую систему управления пакетами они используют и каков соответствующий формат пакетов Linux. Мы поговорим об этом подробнее в главе 3, а пока сосредоточимся на том, как выбрать подходящий дистрибутив Linux в зависимости от потребностей. Прежде чем сделать выбор, стоит немного узнать о наиболее распространенных дистрибутивах: мы обзорно рассмотрим их в следующем разделе.

Распространенные дистрибутивы Linux

Здесь перечислены самые популярные дистрибутивы Linux на момент написания этой книги, а также соответствующие системы управления пакетами. Большинство этих дистрибутивов — свободные платформы с открытым исходным кодом. Если у того или иного дистрибутива есть коммерческая версия, это особо отмечено.

- **Fedora, CentOS Stream и Red Hat Enterprise Linux.** CentOS и производные от нее системы используют систему управления пакетами **RPM (RPM**

Package Manager, ранее **Red Hat Package Manager**). **CentOS Stream**, дистрибутив которой распространяется по модели скользящего выпуска, основана на проекте Fedora с открытым исходным кодом. Эта ОС подходит как для серверов, так и для рабочих станций. **Red Hat Enterprise Linux (RHEL)** — это коммерческая версия, которая основана на CentOS Stream и позиционируется как стабильная платформа с долгосрочной поддержкой. Дистрибутив Rocky Linux поддерживается сообществом, использует исходный код RHEL и бинарно совместим с ней.

- В **Debian** и большинстве производных от него дистрибутивов пакетами управляет система **dpkg (Debian Package)**. Новые выпуски Debian выходят гораздо реже, чем другие основанные на нем дистрибутивы (например, Linux Mint или Ubuntu), но эта система считается более стабильной.
- **Ubuntu** использует системы управления пакетами **APT (Advanced Package Tool)** и **dpkg**. Ubuntu — один из самых популярных дистрибутивов Linux; его новые версии выходят каждые полгода, а более стабильные версии с долгосрочной поддержкой (**LTS, Long Term Support**) — раз в два года.
- **Linux Mint**, который тоже использует APT, основан на Ubuntu и в основном подходит для настольных компьютеров — в частности потому, что потребляет меньше памяти (благодаря среде рабочего стола Cinnamon, в отличие от GNOME, которую использует Ubuntu). Существует также версия Linux Mint, которая базируется на Debian и называется **LMDE (Linux Mint Debian Edition)**.
- **openSUSE** использует такие системы управления пакетами, как **RPM, YaST** и **Zypper**. У openSUSE есть две версии: одна из них называется Tumbleweed и выходит в режиме скользящего выпуска, предлагая новейшие возможности Linux, а другая — Leap, выпускается по регулярному графику и использует ту же кодовую базу, что и SUSE Linux Enterprise. Обе версии подходят и для настольного компьютера, и для сервера. SUSE Linux Enterprise Server — это коммерческая система. Пока не появилась Ubuntu, openSUSE считалась одним из самых удобных дистрибутивов Linux для настольных компьютеров.

В этой книге мы сосредоточимся на дистрибутивах Linux, которые широко используются как в профессиональном сообществе, так и в корпоративной среде — например, **Ubuntu, Fedora/Rocky Linux** и **openSUSE**. Большинство примеров из книги одинаково хорошо работают в любом дистрибутиве Linux. Мы будем указывать, какой дистрибутив сами используем для тех или иных примеров или сценариев.

Теперь, когда вы познакомились с наиболее распространенными дистрибутивами Linux, мы дадим несколько советов о том, как выбрать дистрибутив.

Как выбрать дистрибутив Linux

Выбор дистрибутива Linux зависит от многих факторов, которые вытекают из функциональных требований. Их исчерпывающий анализ выходит за рамки этой главы, но часто бывает достаточно рассмотреть всего несколько основных параметров, чтобы принять правильное решение.

- **Платформа.** Чтобы выбрать дистрибутив Linux, вероятно, в первую очередь стоит понять, для какой платформы он предназначен — серверной, настольной или встраиваемой. Серверные и встраиваемые версии Linux обычно поставляются с основными службами операционной системы и компонентами, которые необходимы для конкретных приложений (сетевых подключений, HTTP, FTP, SSH и электронной почты), а набор этих служб и компонентов формируется в основном из соображений производительности и оптимизации. С другой стороны, в настольных системах Linux обычно предустановлено много программных пакетов, включая графический интерфейс пользователя. У некоторых дистрибутивов Linux (например, **Ubuntu**, **Fedora** и **openSUSE**) есть серверная и настольная версии, но большинство дистрибутивов предлагают минимально работоспособную операционную систему, которую нужно дополнительно настраивать (например, **Rocky Linux** и **Debian**). Обычно такие дистрибутивы хорошо годятся на роль серверных платформ Linux. Наоборот, некоторые дистрибутивы (**elementary OS**, **Pop!_OS** или **Deepin**) разработаны специально для настольных систем. Для встраиваемых платформ существуют высокооптимизированные дистрибутивы Linux, такие как **Raspberry Pi OS** (ранее **Raspbian**) и **OpenWRT**, которые предназначены для устройств небольшого форм-фактора с ограниченными аппаратными ресурсами.
- **Инфраструктура.** Сегодня существует огромное количество сред, где развертываются приложения и серверные платформы, начиная с физического оборудования и локальных центров обработки данных и заканчивая гипервизорами, контейнерами и облачной инфраструктурой. Чтобы оценить, какой дистрибутив Linux подойдет для той или иной среды, следует учитывать ресурсы и затраты. Например, если экземпляр Linux работает на нескольких процессорах и интенсивно потребляет память и другие ресурсы, — скорее всего, он обойдется дороже, если запускать его в облаке или на хостинге **виртуальных выделенных серверов (VPS)**. Облегченные дистрибутивы Linux требуют меньше ресурсов и легче масштабируются в средах с контейнерной рабочей нагрузкой и контейнерными службами (например, Kubernetes и Docker). У большинства дистрибутивов Linux уже есть свои облачные образы, которые доступны для всех основных облачных провайдеров (например, Amazon AWS, Microsoft Azure и GCE). Образы контейнеров Docker для различных дистрибутивов Linux можно загрузить с Docker Hub (hub.docker.com), причем одни образы тяжелее, чем другие. Например, образ Docker для **Ubuntu Server** значительно объемнее, чем для **Alpine Linux**, и это может стать решающим фактором того, какой дис-

трибутив предпочесть. Кроме того, в связи с тем, что в последнее время все больше рабочих процессов и служб переходят на контейнеры, для некоторых дистрибутивов Linux выпускается отдельная версия операционной системы, которая оптимизирована для той или иной инфраструктуры. Например, соответствующие версии дистрибутива Fedora — **Fedora CoreOS** (для контейнерных рабочих процессов) и **Fedora IoT** (для экосистем интернета вещей).

- **Производительность.** Пожалуй, все дистрибутивы Linux можно отрегулировать так, чтобы достичь высоких показателей в тестах производительности процессора, графического процессора, памяти и хранилища. Производительность следует оценивать в тесной связи с платформой и целевым применением. Вряд ли на Raspberry Pi будет эффективно работать сервер электронной почты, зато сервер потоковой передачи мультимедиа проявит себя прекрасно (если подключить внешнее хранилище). Также учитывайте затраты на тонкую настройку производительности. Серверные и настольные версии **Rocky Linux, Debian, openSUSE, Fedora** и **Ubuntu** изначально довольно неплохо оптимизированы для типичных сценариев использования. Серверные версии можно легко настроить под конкретное приложение или службу, ограничив набор программ только теми, которые необходимы для работы приложения. Стремясь еще больше повысить производительность, некоторые специалисты даже перекомпилируют тот или иной облегченный дистрибутив Linux (например, **Gentoo**), чтобы воспользоваться оптимизацией ядра на уровне компилятора для определенных подсистем (например, сетевого стека или прав доступа). Как и в случае многих других параметров, если вы выбираете дистрибутив Linux из соображений производительности того или иного приложения или платформы, вы рискуете напрасно потратить время и силы на тонкую настройку, потому что чаще всего обычные дистрибутивы Linux работают вполне эффективно.
- **Безопасность.** Когда речь идет о безопасности, не забывайте, что система надежна лишь настолько, насколько надежно ее самое слабое звено. Одно небезопасное приложение или один «дырявый» системный компонент подвергает риску всю систему. Поэтому имеет смысл тщательно проанализировать безопасность дистрибутива Linux в контексте целевых приложений и среды развертывания. Например, можно говорить о *безопасности настольной системы*, если дистрибутив Linux служит рабочей станцией, с помощью которой вы посещаете сайты, загружаете медиафайлы, устанавливаете различные программные пакеты и запускаете приложения. То, насколько безопасно выполняются все эти операции (в смысле защиты от вредоносных программ, вирусов и вторжений), может служить хорошим показателем безопасности системы. Существуют дистрибутивы Linux, которые специализируются на безопасности и изоляции приложений и хорошо подходят для настольных систем: **Qubes OS, Kali Linux, Whonix, Tails** и **ParrotOS Security Edition**. Некоторые из этих дистрибутивов были разработаны для тестирования на проникновение и исследований в области безопасности.

С другой стороны, можно рассмотреть безопасность дистрибутивов Linux *как серверных систем*. В этом случае, если вы регулярно обновляете операционную систему пакетами и компонентами, то она уже будет достаточно хорошо защищена. Чтобы дополнительно уменьшить потенциальную поверхность атаки, можно удалить неиспользуемые сетевые службы и настроить более строгие правила межсетевого экрана. Большинство дистрибутивов Linux оснащены необходимыми инструментами и службами для такой регуляции. Как правило, первое условие безопасности платформы состоит в том, чтобы выбрать дистрибутив с *частыми* и *стабильными* обновлениями или циклами выпуска (например, **Rocky Linux**, **RHEL**, **Ubuntu LTS** или **SUSE Linux Enterprise**).

- **Надежность.** Если новые версии дистрибутива Linux выпускаются часто и в каждой версии добавляется много нового кода, — такой дистрибутив обычно менее надежен. Для подобных дистрибутивов важно выбрать *стабильную* версию. Например, **Fedora** обновляется очень часто и развивается быстрее, чем большинство систем семейства Linux. Тем не менее не стоит потакать мифам о том, будто Fedora или другие быстро развивающиеся дистрибутивы Linux, такие как openSUSE Tumbleweed, проигрывают по надежности. Не забывайте, что некоторые из самых надежных дистрибутивов Linux — **RHEL** и **SUSE Linux Enterprise** — основаны на Fedora и openSUSE соответственно.

Не существует волшебной формулы, которая помогала бы выбрать дистрибутив Linux. Чаще всего выбор зависит от целевой платформы (сервер, настольный компьютер или интернет вещей), а также от ваших личных предпочтений. В средах реальной эксплуатации большинство перечисленных факторов становятся критическими, так что фактически выбор платформы Linux сводится к нескольким вариантам, которые лучше всего зарекомендовали себя на практике.

Теперь, когда вы представляете себе, что такое дистрибутив Linux, а также знаете о том, какие дистрибутивы наиболее распространены и для каких ситуаций подходит каждый из них, в следующих двух разделах мы расскажем об основах и продвинутых приемах установки Linux.

Устанавливаем Linux: первые шаги

Этот раздел — краткое руководство по элементарной установке Linux независимо от дистрибутива. Для практических примеров и специфических рекомендаций мы используем Ubuntu. Мы также обзорно рассмотрим различные среды, в которых устанавливается Linux. Сейчас усиливается тенденция к тому, чтобы создавать гибридные облачные инфраструктуры, где локальные центры обработки данных сочетаются с облачными развертываниями, а Linux работает либо на физическом оборудовании, либо под управлением гипервизора, либо в виртуальной машине, либо в контейнере Docker.

В большинстве случаев Linux устанавливается по одним и тем же принципам. О том, как развертывать Linux в контейнерах Docker, см. главу 13.

В следующих разделах мы продемонстрируем, как установить Linux на физическое оборудование, а также в управляющей системе Windows 11 с помощью различных гипервизоров виртуальных машин и WSL¹. В macOS установка практически не отличается от установки в Windows с помощью гипервизора, поэтому мы не будем рассматривать ее отдельно.

Как установить Linux на физическое оборудование

В этом разделе описаны основные шаги, которые необходимы, чтобы установить Linux на **физическое оборудование** — ноутбуки, настольные компьютеры, рабочие станции и серверы. В общих чертах ключевые этапы — загрузить дистрибутив в виде образа ISO, создать загрузочный носитель, опробовать портативный режим² и, наконец, установить систему.

В следующих разделах вы увидите, что процедура, которая здесь описана, в равной степени применима и для того, чтобы установить Linux в виртуальной машине.

Шаг 1. Загрузить образ ISO

Прежде всего загрузим выбранный дистрибутив Linux. Большинство дистрибутивов обычно доступны в формате ISO на официальных сайтах. Например, Ubuntu Desktop можно загрузить по адресу ubuntu.com/download/desktop.

На следующем этапе с помощью образа ISO можно создать загрузочный носитель, который нужен, чтобы установить Linux. Также образ ISO позволяет установить Linux на виртуальной машине, как будет показано в следующем разделе.

Шаг 2. Создать загрузочный носитель

Когда Linux устанавливается на настольный компьютер или рабочую станцию, загрузочный носитель обычно представляет собой диск CD/DVD или устройство USB. Если под рукой есть пишущий привод DVD, можно просто записать дистрибутив в формате ISO на диск DVD. Но поскольку современные компьютеры,

¹ Windows Subsystem for Linux (WSL) — это технология, позволяющая запускать приложения и инструменты Linux непосредственно в Windows без виртуальной машины. WSL обеспечивает совместимость с большинством команд и программ Linux, что удобно для разработчиков и тех, кто работает с обеими системами на одном компьютере. — *Примеч. ред.*

² В данном случае имеется в виду Live-режим — режим работы Linux без необходимости установки. Такой режим необходим для проверки совместимости дистрибутива с оборудованием, а также оценки возможностей и внешнего вида приложений. — *Примеч. ред.*

особенно ноутбуки, редко оснащаются оптическими приводами, в качестве загрузочных носителей чаще используются накопители USB.

Есть и третий вариант: использовать загрузочный сервер **PXE (Preboot eXecution Environment)**, то есть «предзагрузочная среда выполнения»). PXE — это клиент-серверная среда, в которой клиент с поддержкой PXE загружает пакет программного обеспечения по локальной или глобальной сети с сервера PXE и запускает этот пакет. При этом не нужны физические загрузочные устройства (CD/DVD, USB) и снижаются накладные расходы на установку, особенно если у вас много клиентов и операционных систем. Внутреннее устройство PXE выходит за рамки этой книги, но в конце этой главы мы дадим краткое представление о том, как с помощью PXE установить Linux. Чтобы больше узнать о PXE, в качестве отправной точки можно ознакомиться со статьей en.wikipedia.org/wiki/Preboot_Execution_Environment.

Чтобы относительно просто создать загрузочный носитель USB с тем или иным дистрибутивом Linux, можно использовать такие инструменты, как **UNetbootin** (unetbootin.github.io) или **balenaEtcher** (etcher.balena.io). Обе эти утилиты — кросс-платформенные: они работают под Windows, Linux и macOS.

Чтобы создать загрузочный носитель USB в Windows, в этом примере мы будем использовать balenaEtcher:

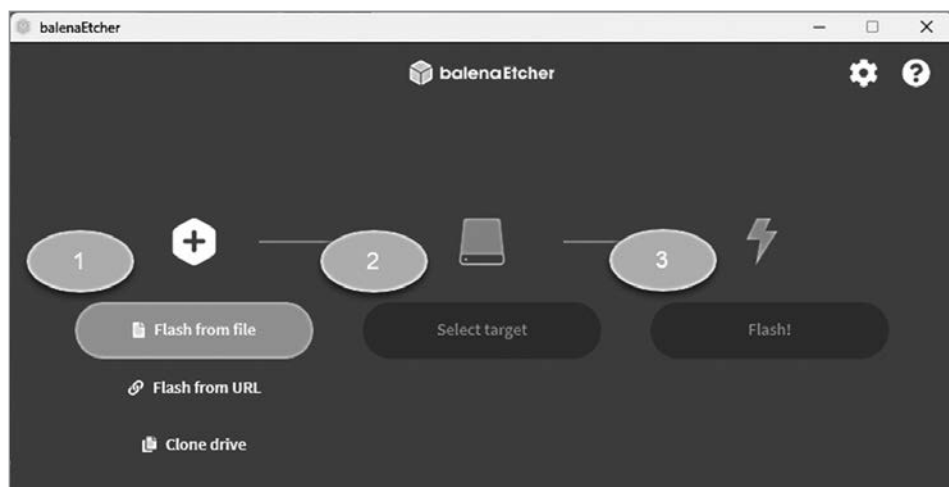


Рис. 1.2. Создание загрузочного носителя USB с помощью balenaEtcher¹

¹ Flash from file — Загрузить прошивку из файла; Flash from URL — Загрузить прошивку из URL; Clone drive — Клонировать носитель; Select target — Выбрать целевое устройство; Flash! — Применить прошивку.

Вот как можно создать загрузочный носитель USB с дистрибутивом Ubuntu Desktop с помощью balenaEtcher. (Допустим, что вы уже загрузили образ Ubuntu Desktop и установили Etcher — в нашем случае на Windows 11.)

1. Выберите файл ISO с нужным дистрибутивом Linux.
2. Выберите целевой носитель USB.
3. Прошейте выбранный носитель с помощью выбранного файла ISO.

Вся процедура займет пару минут, и загрузочный носитель USB будет готов. Теперь давайте посмотрим, как его можно использовать.

Шаг 3. Попробовать загрузочный носитель в портативном режиме

Этот шаг необязателен.

Для большинства дистрибутивов Linux можно загрузить образ ISO, который запускается с портативного носителя. Не все дистрибутивы предлагают такую возможность (по крайней мере по умолчанию), однако, например, для **Ubuntu** и **Fedora** поставляются официальные портативные загрузчики.

Если у вас есть загрузочный носитель с выбранным дистрибутивом Linux, можно запустить операционную систему в портативном режиме, не устанавливая ее. Другими словами, можно опробовать и протестировать дистрибутив Linux, прежде чем решить, устанавливать его или нет. Портативный Linux загружается в оперативную память компьютера, не используя дисковое хранилище. Перед этим нужно убедиться, что на компьютере хватит оперативной памяти, чтобы помимо самого дистрибутива Linux вместить минимальный объем памяти, который ему необходим.

Загружая компьютер с носителя, проверьте, что порядок загрузки в BIOS задан так, чтобы носитель читался с наивысшим приоритетом. На Mac для этого нужно сразу после звукового сигнала запуска нажать клавишу **Option** и выбрать загрузочный носитель USB. На компьютере с Windows нужно зайти в интерфейс BIOS и выбрать соответствующее загрузочное устройство; в зависимости от материнской платы для этого обычно нужно нажать клавишу **Esc** или одну из клавиш **F1**, **F2**, **F10** или **F12**. В некоторых особых случаях для этого может служить другая клавиша (чаще всего одна из функциональных). В нижней части экрана начальной загрузки обычно указывается, какие клавиши нужно нажать.

После перезагрузки мастер установки дистрибутива Linux предложит запустить систему в портативном режиме, как показано на иллюстрации для Ubuntu Desktop (рис. 1.3).

Далее посмотрим, как установить наш дистрибутив Linux с помощью загрузочного носителя.

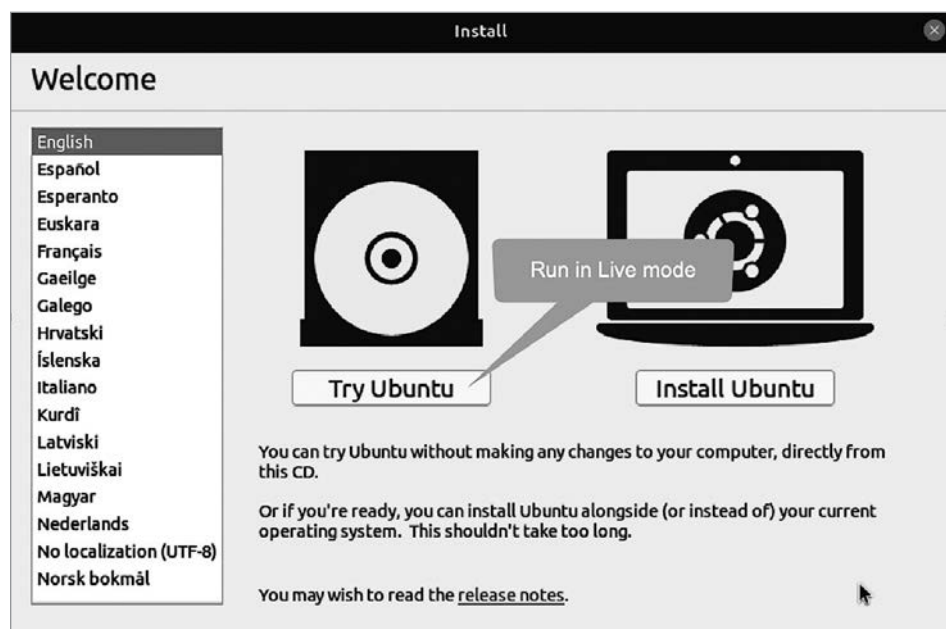


Рис. 1.3. Выбор портативного режима для Ubuntu Desktop

Шаг 4. Выполнить установку

Чтобы начать установку дистрибутива Linux, загрузите компьютер с загрузочного носителя, который вы создали на шаге 2. Чтобы система могла загрузиться с внешнего устройства, иногда требуется изменить порядок загрузки в BIOS, особенно если используется загрузочный накопитель USB. Чтобы выбрать нужное загрузочное устройство, сделайте все, как указано в предыдущем пункте.

В следующих разделах мы продемонстрируем, как установить Ubuntu с помощью образа ISO. Мы рассмотрим настольную и серверную версии дистрибутива и обсудим, чем они различаются. В отличие от Ubuntu, Rocky Linux и CentOS Stream не делятся на настольную и серверную версии: по сути это серверные платформы с необязательным графическим интерфейсом. Похожим образом openSUSE предлагает единый установщик как для настольных компьютеров, так и для серверов, в то время как у Fedora предусмотрены отдельные установочные комплекты для каждой из этих сред.

Далее мы расскажем о том, как установить Linux в виртуальной машине.

Как установить Linux в виртуальной машине

В этом разделе мы предлагаем краткое руководство о том, как подготовить среду виртуальной машины для установки Linux под управлением различных гипервизоров.

Виртуальная машина — это изолированная программная абстракция физического компьютера. Виртуальные машины развертываются поверх **гипервизора**, который обеспечивает среду выполнения для машины и управляет ее ресурсами. Вот некоторые популярные гипервизоры общего назначения:

- **Oracle VirtualBox** ([virtualbox.org](https://www.virtualbox.org/));
- **VMware Workstation** ([vmware.com/products/desktop-hypervisor/workstation-and-fusion](https://www.vmware.com/products/desktop-hypervisor/workstation-and-fusion/));
- **Hyper-V** (learn.microsoft.com/virtualization/hyper-v-on-windows/about).

Первые два гипервизора — кроссплатформенные приложения для виртуализации, которые работают на процессорах Intel и AMD под управлением Windows, macOS и Linux. Hyper-V доступен только для Windows Pro, Enterprise или Education (версия 10 или выше).

На момент написания этой книги гипервизоры для компьютеров Mac с процессором Apple silicon предоставляют только VMware Player и Parallels¹. Oracle VirtualBox для архитектуры Arm пока находится в предварительной версии². Решения от VMware и Parallels — платные программы для macOS.

В виртуальной машине Linux устанавливается практически так же, как на физическом оборудовании, хотя важно обратить внимание на оснащение и конфигурацию виртуальной машины и убедиться, что она удовлетворяет минимальным системным требованиям нужного дистрибутива Linux. В следующих разделах мы установим Ubuntu на VMware Workstation под Windows.

Имейте в виду, что установка Linux на VMware Player под macOS происходит практически так же, и мы не будем дублировать соответствующий процесс в этом издании книги. Особенности macOS обсуждались в первом издании, но здесь мы решили их опустить, учитывая ограниченную доступность гипервизоров для платформы Apple silicon. Если вас интересует, как установить Linux на оборудовании Apple, обратите внимание на проект Asahi Linux (asahilinux.org), цель которого — создать полнофункциональный дистрибутив Linux для компьютеров на основе Apple silicon.

В следующем разделе мы обзорно рассмотрим установку Ubuntu Server LTS. Если вы хотите установить Ubuntu в виртуальной машине, сначала понадобится выполнить несколько шагов, чтобы оснастить эту машину. Если вы не ис-

¹ Однако это не ограничивает пользователя в установке QEMU и производных продуктов. Например, таких как UTM или OrbStack. — *Примеч. ред.*

² Начиная с версии VirtualBox 7.1 (сентябрь 2024) управляющие системы на основе архитектуры Arm поддерживаются с определенными ограничениями (см. [virtualbox.org/manual/topics/Introduction.html#arm-host-limitations](https://www.virtualbox.org/manual/topics/Introduction.html#arm-host-limitations)). — *Примеч. ред.*

пользуете виртуальную машину, переходите сразу к разделу «Процедура установки».

Как оснастить виртуальную машину с помощью VMware Workstation

В этом разделе мы создадим виртуальную машину для Ubuntu Server с помощью VMware Workstation под Windows 11. На момент написания этой книги самая свежая версия — VMware Workstation 17, которая доступна как для бесплатно-го, так и для коммерческого использования¹.

1. После запуска гипервизора первым делом нажмите на значок **Create a New Virtual Machine** (Создать новую виртуальную машину). Откроется новое окно с мастером создания виртуальной машины, где можно выбрать образ ISO того дистрибутива Linux, который вы хотите установить².
2. Нажмите кнопку **Browse** (Обзор), а затем откройте образ из локального или сетевого расположения.
3. Нажмите кнопку **Next** (Далее), после чего нужно будет задать имя новой виртуальной машины и указать ее местоположение на диске. Мы оставим путь, который гипервизор предлагает по умолчанию, и назовем машину **Ubuntu**³.
4. Нажмите **Next**. В следующем окне нужно указать максимальный размер диска для виртуальной машины. По умолчанию задано 20 Гбайт — рекомендуемый размер для Ubuntu Server, который мы не будем менять.
5. Если снова нажать **Next**, появится окно с параметрами виртуальной машины. По умолчанию гипервизор предоставляет ей 2 ядра процессора и 4 Гбайт оперативной памяти. Можно нажать кнопку **Customize Hardware** (Настроить оборудование), чтобы изменить стандартные настройки, если ваше оборудование это позволяет. Как правило, мы рекомендуем, чтобы в управляющей системе было не менее 16 Гбайт оперативной памяти и хотя бы 8-ядерный процессор — это позволяет создавать виртуальные машины разумной мощности. Когда вы закончите настраивать оборудование, нажмите кнопку **Close** (Заккрыть) в правой нижней части окна. Вы вернетесь в главное окно мастера.

¹ С 2024 года гипервизоры Workstation и Fusion семейства VMware стали бесплатными: с мая 2024-го — для персонального использования, а с ноября 2024-го — также и для коммерческого. — *Примеч. ред.*

² Первое окно мастера предлагает выбрать режим установки: **Typical** (Стандартная) или **Custom** (Настраиваемая). Если выбрать **Typical**, то в следующем окне можно указать путь к файлу дистрибутива в формате ISO. — *Примеч. ред.*

³ На момент подготовки этого русскоязычного издания самой свежей версией с долгосрочной поддержкой (LTS) является Ubuntu Server 24.04.1 (август 2024). Процедура установки фактически не отличается от версии 22.04.1, которая описана в книге, и большинство последующих примеров из книги работают одинаково в обеих системах. — *Примеч. ред.*

6. Нажмите **Finish (Завершить)**, чтобы закончить настройку, создать виртуальную машину и инициализировать ее. На следующем снимке экрана показано, как выглядит только что созданная машина, которая запущена под управлением VMware Workstation.

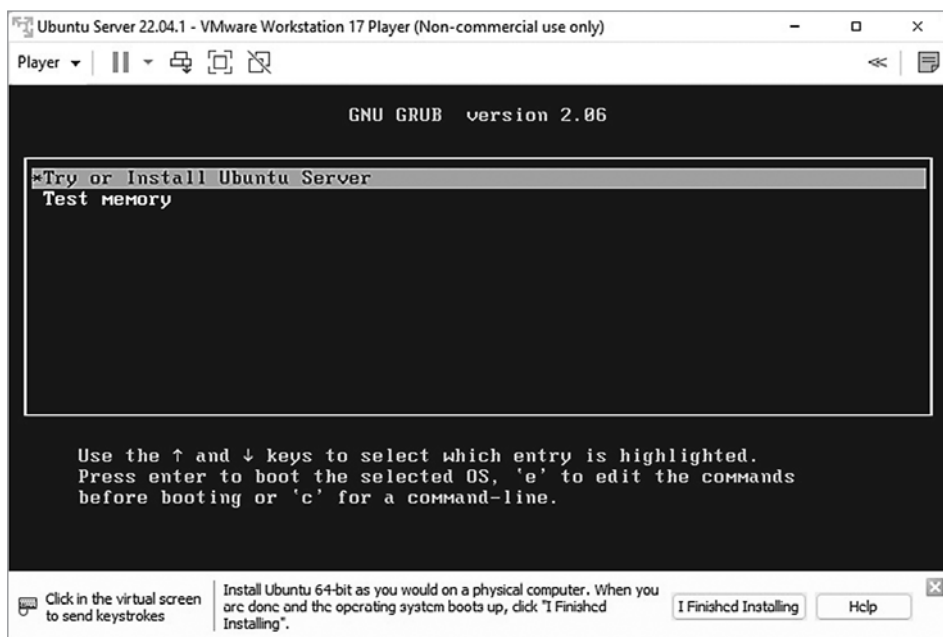


Рис. 1.4. Начальный экран установки Linux после запуска новой виртуальной машины

Процедура установки

Вот как выглядит обычный процесс установки Ubuntu Server LTS после того, как виртуальная машина загружается в режиме установки:

1. На экране приветствия предлагается указать **язык**. Выберите нужный язык и нажмите **Enter**.
2. Мастер установки может предложить вам *обновить программу установки*, если обновление доступно. Можно обновить программу или продолжить без обновления; мы выбираем первый вариант.
3. На следующем экране можно выбрать **раскладку клавиатуры**. В нашем примере будет фигурировать английская раскладка, поэтому выберите в списке **English (US)**, затем **Done (Готово)** и нажмите **Enter**¹.

¹ Если выбрать раскладку, отличную от английской, программа установки предложит также задать сочетание клавиш для переключения раскладки (например, **Alt + Shift** или **Caps Lock**). — *Примеч. ред.*

4. На следующем экране нужно выбрать **базовую конфигурацию установки** из следующих вариантов: **Ubuntu Server** и **Ubuntu Server (minimized)** (минимальный). Можно также включить поиск **драйверов сторонних производителей** (**Search for third-party drivers**). Мы выбираем **Ubuntu Server** и отмечаем поиск драйверов. Чтобы перемещаться между параметрами на экране, нажимайте клавишу **Tab** или клавиши со стрелками. Чтобы выбрать вариант, нажмите клавишу пробела. Отметив варианты, выберите **Done** и нажмите **Enter**.
5. На следующем экране показана **конфигурация сети** (**Network configuration**). Если конфигурация по умолчанию вам подходит, нажмите **Enter**, чтобы перейти к следующему экрану.
6. Мастер установки предложит настроить **прокси-сервер** (**Proxy configuration**). Если вам это не нужно, просто нажмите **Enter**, чтобы перейти далее.
7. Теперь мастер предложит настроить **зеркало архивных репозитория Ubuntu** (**Ubuntu archive mirror configuration**) по умолчанию. Выберите зеркало в зависимости от того, где вы находитесь, или просто оставьте стандартный адрес. Нажмите **Enter**.
8. На следующем экране можно настроить **хранилище и сегментацию дисков** (**Guided storage configuration**). Мы будем использовать весь 20-гигабайтный диск с настройками по умолчанию, поэтому выберите **Done** и нажмите **Enter**.
9. Появится **сводка конфигурации хранилища** (**Storage Configuration**). Если вас все в ней устраивает, просто нажмите **Enter**.
10. Появится предупреждение **Confirm destructive action** (Подтвердите необратимое действие) с вопросом о том, хотите ли вы продолжить установку с текущими настройками. Выберите **Continue** и нажмите **Enter**.
11. На следующем экране нужно ввести сведения вашей **учетной записи** (**Profile configuration**): ваше имя, имя сервера, имя пользователя и пароль. Задайте их и перейдите к следующему экрану.
12. Мастер установки предложит выбрать, устанавливать ли **сервер OpenSSH** (**SSH configuration**). Выберите **Install OpenSSH** (Установить OpenSSH). Если у вас есть ключи SSH, которые вы хотите импортировать, их можно указать здесь. После этого перейдите к следующему экрану.
13. Затем можно выбрать и установить те или иные пакеты **Snap** (**Featured server snaps**) для нового сервера. В зависимости от ваших нужд можно установить их сейчас или позже (в ручном режиме). Среди предлагаемых пакетов — **docker**, **microk8s**, **powershell**, **nextcloud** и **livepatch**. Выберите нужные пакеты Snap и перейдите к следующему экрану.
14. Начнется **процесс установки** (**Installing system**), который может занять несколько минут. Наберитесь терпения и дождитесь, пока появится приглашение перезагрузить компьютер после установки операционной системы (**Reboot Now**).

После перезагрузки появится экран входа в систему, и вы сможете использовать новую виртуальную машину с Ubuntu Server под управлением VMware Workstation на Windows 11. Вот мы и установили серверный дистрибутив Ubuntu.

Большинство других дистрибутивов устанавливается примерно так же, как Ubuntu. При установке настольной версии обычно доступен графический интерфейс. В предыдущем примере мы устанавливали серверную операционную систему, поэтому графического интерфейса не было, а был только минималистичный текстовый интерфейс.

Мы не будем подробно разбирать установку других дистрибутивов, но продемонстрируем интерфейс установки Rocky Linux:

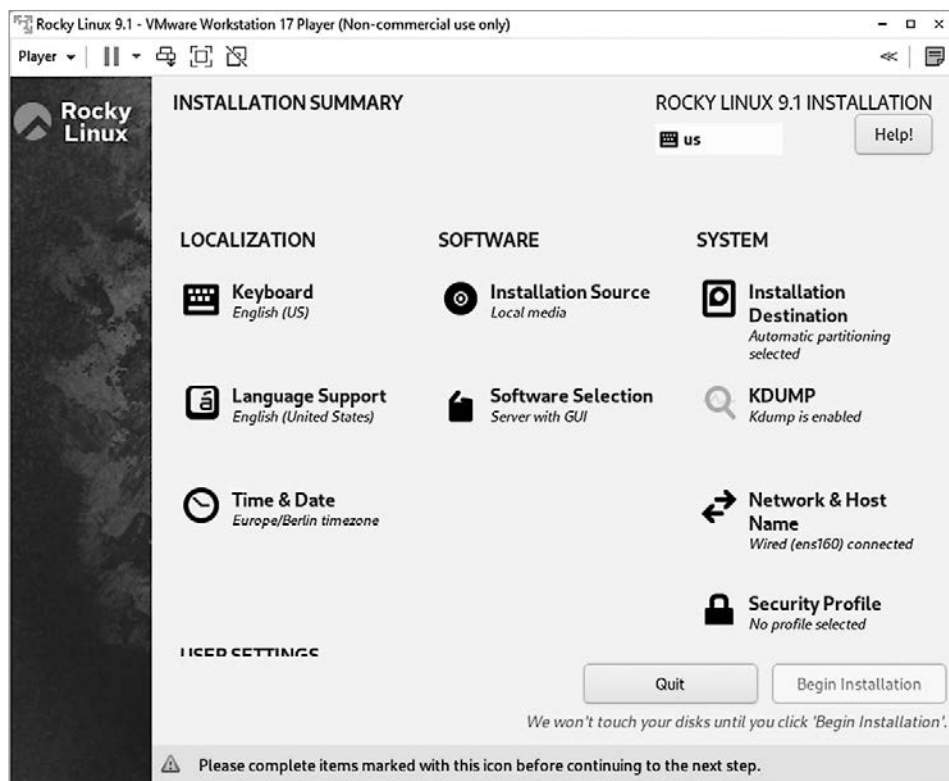


Рис. 1.5. Графический интерфейс установки Rocky Linux

К этому моменту мы научились выполнять элементарную установку Linux. Попутно мы создали загрузочный носитель USB, с помощью которого Linux чаще всего устанавливается на компьютер. Кроме того, мы вкратце поговорили о том, как настроить виртуальную машину для Linux с помощью гипервизора VMware Workstation для Windows 11.

Далее в этой главе вы узнаете, как установить и запустить дистрибутив Linux на платформе Windows без отдельного гипервизора — с помощью подсистемы Windows для Linux (WSL).

Как оснастить виртуальную машину с помощью Hyper-V

В этом разделе мы создадим виртуальную машину для Ubuntu Server с помощью решения Microsoft Hyper-V, которое доступно в Windows 11 Pro¹.

Прежде всего нужно будет включить гипервизор Hyper-V, потому что по умолчанию он отключен. Для этого откройте диалоговое окно **Компоненты Windows (Windows Features)** и установите флажок **Hyper-V**, как показано на следующем рисунке. После этого понадобится перезагрузить компьютер.

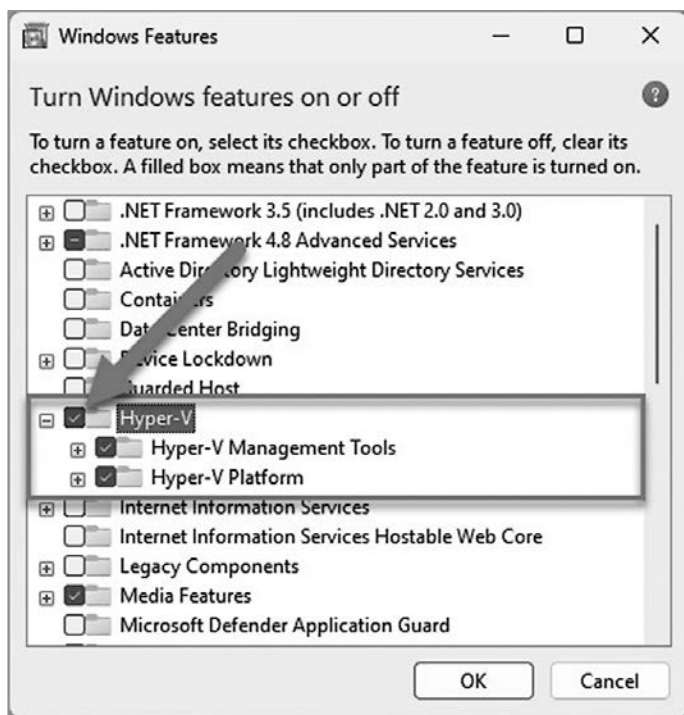


Рис. 1.6. Как включить Hyper-V в Windows 11 Pro

¹ Гипервизор Hyper-V был впервые выпущен в 2008 году и впоследствии стал необязательным компонентом Windows 8 и более поздних клиентских версий Windows (вместо Windows Virtual PC), а также Windows Server начиная с версии 2012. — *Примеч. ред.*

Чтобы создать новую виртуальную машину, необходимо запустить **диспетчер Hyper-V (Hyper-V Manager)**. Его интерфейс состоит из трех панелей. В правой панели **Actions (Действия)** находится пункт **New (Создать)**¹. Щелкните по нему и выберите вариант **Virtual Machine ... (Виртуальная машина...)**. Откроется новое окно, в котором можно настроить новую виртуальную машину с помощью таких действий:

1. Задайте имя и расположение виртуальной машины. В этом примере мы назовем ее **Ubuntu** и оставим расположение по умолчанию. Нажмите **Next**.
2. Задайте поколение виртуальной машины. Hyper-V предлагает два варианта: **Generation 1 (Поколение 1)** и **Generation 2**. Вторым вариантом виртуализируется UEFI и подходит, в частности, для сетевой установки (PXE). Выберите **Generation 1** и нажмите **Next**.
3. Укажите объем оперативной памяти. По умолчанию задано минимальное значение 4 Гбайт, при этом выбрана опция динамической памяти. Оставьте эти стандартные настройки и нажмите **Next**.
4. Настройте сеть, выбрав из выпадающего списка **Default Switch (Коммутатор по умолчанию)**. Нажмите **Next**.
5. Настройте виртуальный жесткий диск, задав его объем и расположение. Нажмите **Next**.
6. Следующее окно предлагает установить операционную систему сейчас или позже. Выберите образ **ISO Ubuntu Desktop** из нужного расположения и нажмите **Next**.
7. В следующем окне отображается сводка конфигурации виртуальной машины. Если нажать **Back (Назад)**, можно изменить любые из этих параметров. Закончив настройку, нажмите **Finish**, и виртуальная машина будет создана.

На рис. 1.7 показана новая виртуальная машина **Ubuntu**, которая запущена под управлением **Hyper-V**:

После этого дистрибутив можно установить так же, как в предыдущем разделе, поэтому мы не будем повторять здесь всю процедуру. В следующем разделе мы рассмотрим другой гипервизор, на этот раз от Oracle.

¹ Если вы впервые работаете с **Hyper-V**, перед этим может еще понадобится выбрать на правой панели **Подключиться к серверу... (Connect to Server...)**, а затем **Локальный компьютер (Local computer)**. — *Примеч. ред.*

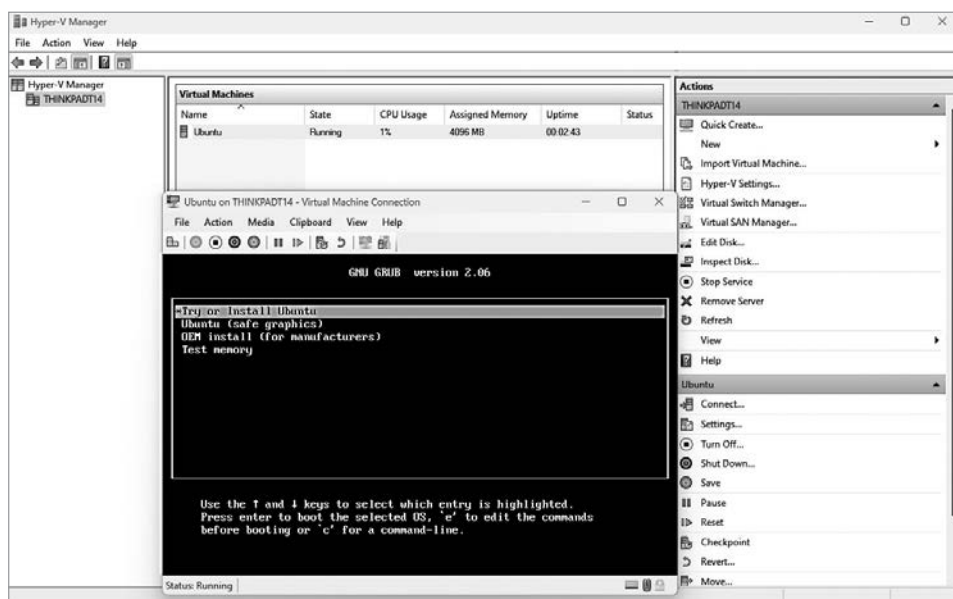


Рис. 1.7. Новая виртуальная машина Ubuntu под управлением Hyper-V

Как оснастить виртуальную машину с помощью Oracle VirtualBox

Oracle VirtualBox — это бесплатное мультиплатформенное программное обеспечение, которое доступно для Windows, macOS и Linux. Мы покажем, как создать виртуальную машину для Linux на Windows 11 (предполагается, что у вас уже установлен VirtualBox). Гипервизор от Oracle позволяет оснастить виртуальную машину с помощью удобного интерфейса, и сейчас мы рассмотрим эту процедуру. В этом примере мы используем дистрибутив Fedora Workstation:

1. Нажмите **New**, чтобы начать создавать новую виртуальную машину. Откроется новое окно, в котором нужно указать ее имя, тип операционной системы и расположение образа ISO. По умолчанию это новое окно находится в **интерактивном режиме (Guided Mode)**, но в нижней части окна можно выбрать **экспертный режим (Expert Mode)**, который даст вам больше контроля над созданием виртуальной машины.
2. Укажите всю необходимую информацию. В нашем случае мы будем использовать Fedora, поэтому так и назовем виртуальную машину. Укажите путь к файлу ISO, и тип операционной системы изменится автоматически. Если вы находитесь в **экспертном режиме**, то увидите также разделы параметров **Unattended Install (Автоматическая установка)**, **Hardware (Оборудование)** и **Hard Disk (Жесткий диск)**, которые по умолчанию скрыты.

3. Поскольку мы устанавливаем Fedora, раздел **Unattended Install** будет неактивен (см. рис. 1.11), поскольку эта функция поддерживается лишь некоторыми операционными системами (Ubuntu, RHEL, Oracle Linux и Windows).
4. В разделе **Hardware** задайте объем основной памяти и количество процессоров для новой машины. Выбирайте параметры в зависимости от своих аппаратных ресурсов, но не забывайте, что они должны удовлетворять минимальным требованиям целевой операционной системы. Здесь мы зададим 4 Гбайт оперативной памяти и два виртуальных процессора.
5. В разделе **Hard Disk** задайте объем дискового пространства для виртуальной машины. Опять же, исходите из своих ресурсов и помните о минимальных системных требованиях. На наш взгляд, под диск следует выделить не менее 20 Гбайт. Выберите местоположение виртуального жесткого диска и нажмите **Finish**.
6. Виртуальная машина будет создана, и вы вернетесь в главное окно VirtualBox, где отображаются сведения о виртуальной машине. Чтобы запустить ее, просто нажмите кнопку **Start** с большой зеленой стрелкой.
7. Появится новое окно с виртуальной машиной.

На следующем снимке экрана показано окно создания виртуальной машины в VirtualBox:

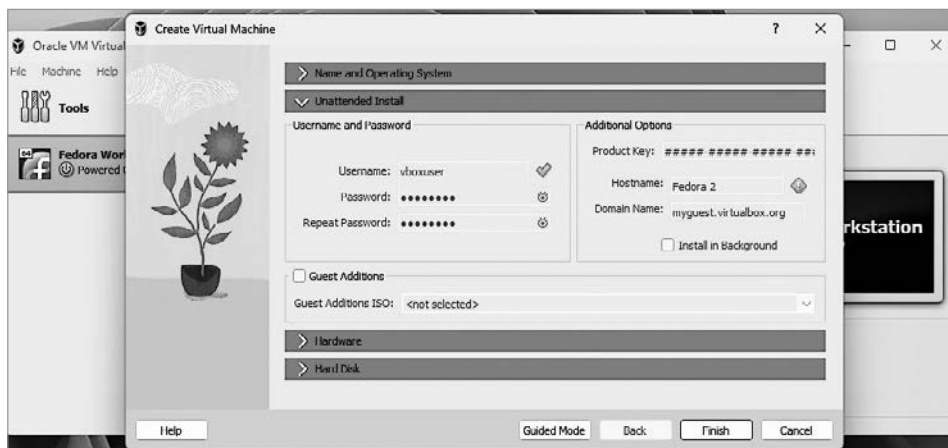


Рис. 1.8. Интерфейс VirtualBox

Как видите, с помощью всех трех основных доступных гипервизоров (от VMware, Microsoft и Oracle) создавать виртуальные машины для Linux очень просто. Сама ОС Linux во всех виртуальных машинах устанавливается одинаково.

В Windows можно не только создавать виртуальные машины, но и запускать Linux относительно новым способом — с помощью подсистемы Windows для Linux. В следующем разделе мы рассмотрим, как это делается.

Как работать с подсистемой Windows для Linux (WSL)

Разработчикам программного обеспечения и системным администраторам часто бывает сложно выбрать подходящее оборудование и операционную систему в соответствии с требованиями своей работы или программного окружения. Специалистов, которые работали на Windows, раньше часто расстраивало, что некоторые стандартные инструменты разработки, фреймворки или серверные компоненты доступны в Linux или macOS, но не поддерживаются в Windows. **Подсистема Windows для Linux (WSL, Windows Subsystem for Linux)** пытается устранить этот пробел.

WSL — это компонент платформы Windows, который предоставляет встроенную среду выполнения GNU/Linux, функционирующую вместе с окружением рабочего стола Windows в версиях Windows 10 и 11. WSL позволяет гладко развертывать и интегрировать некоторые дистрибутивы Linux поверх ядра Windows, не используя специального гипервизора. С помощью WSL можно легко установить и запустить Linux как обычное приложение Windows.

Без WSL дистрибутив Linux можно было развернуть и запустить на платформе Windows только с помощью отдельного гипервизора, такого как Hyper-V, Oracle VirtualBox или VMware Workstation. С WSL выделенный гипервизор не нужен. На момент написания этой книги WSL представляет собой расширение ядра Windows со встроенным гипервизором.

В этом разделе мы покажем, как настроить WSL и запустить дистрибутив Ubuntu под Windows. Начиная с Windows 11 версии 21H2 и Windows 10 версии 21H2, WSL доступна по умолчанию в Microsoft Store, поэтому ее можно установить и настроить без командной строки. Перейдите в Microsoft Store и введите в строку поиска WSL. Из появившегося списка выберите приложение, которое показано на рис. 1.9.

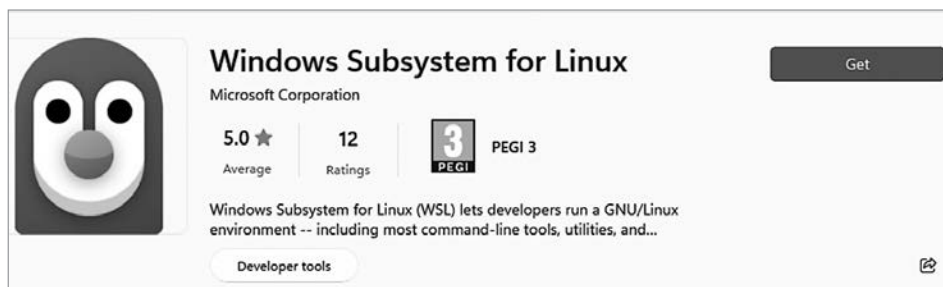
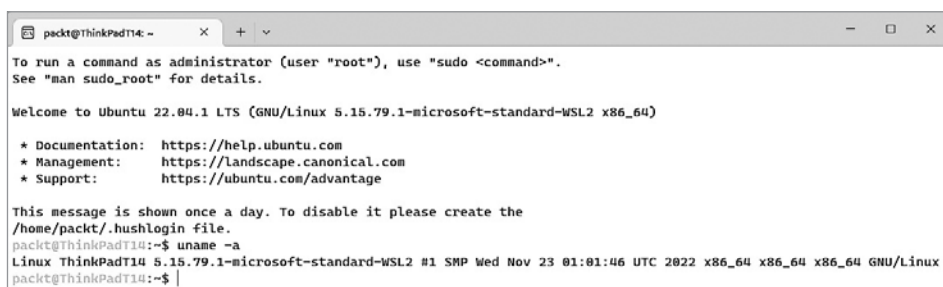


Рис. 1.9. Приложение WSL из Microsoft Store

После того как вы установите WSL, останется только установить любой из дистрибутивов Linux, которые для нее доступны. Если вы откроете свежее установленное приложение WSL, то увидите сообщение терминала о том, что в WSL нет установленных дистрибутивов. Это значит, что нужно вернуться в Microsoft Store, найти нужный дистрибутив и установить его. Например, если искать Linux, вы получите такие результаты, как SUSE Linux Enterprise Server, Oracle Linux, Kali Linux, Ubuntu LTS, Debian, openSUSE Leap и др.

Убедитесь, что в Windows включена служба Hyper-V, потому что именно она отвечает за запуск WSL¹. Чтобы включить ее, откройте диалоговое окно Windows Features (Компоненты Windows) и отметьте Hyper-V, а затем нажмите OK. После установки необходимых компонентов потребуется перезагрузить систему. Hyper-V доступен по умолчанию в Windows 11 Pro, Enterprise и Education, но *не* в версии Home.

Теперь можно установить дистрибутив Linux из Microsoft Store; для примера мы установим Ubuntu. После установки можно открыть терминал, создать пользователя и начать работать в командной строке — все настолько просто. Чтобы запустить установленный дистрибутив, введите его название (например, Ubuntu) в строку поиска Windows, нажмите Enter, и новое окно с командной оболочкой Linux откроется прямо в терминале Windows, как показано на следующем снимке экрана:



```
packt@ThinkPadT14: ~
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

Welcome to Ubuntu 22.04.1 LTS (GNU/Linux 5.15.79.1-microsoft-standard-WSL2 x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

This message is shown once a day. To disable it please create the
/home/packt/.hushlogin file.
packt@ThinkPadT14:~$ uname -a
Linux ThinkPadT14 5.15.79.1-microsoft-standard-WSL2 #1 SMP Wed Nov 23 01:01:46 UTC 2022 x86_64 x86_64 x86_64 GNU/Linux
packt@ThinkPadT14:~$
```

Рис. 1.10. Ubuntu, запущенная в терминале Windows с помощью WSL

¹ На момент подготовки этого издания (2025 г.) WSL 2 работает в любой версии Windows (включая Home) и не требует Hyper-V, а использует только Платформу виртуальных машин (Virtual Machine Platform). Начиная с обновления 20H1 для Windows 10 (май 2020 г.), Microsoft официально рекомендует работать с WSL преимущественно из командной строки. В частности, чтобы установить WSL, можно запустить команду `wsl --install`. Другие команды по управлению WSL можно найти на официальном сайте: learn.microsoft.com/windows/wsl. — *Примеч. ред.*

Более того, при этом вы получите доступ к файловой системе дистрибутива прямо из Проводника Windows. На следующем снимке экрана показана файловая система Ubuntu, к которой можно обращаться из Проводника:

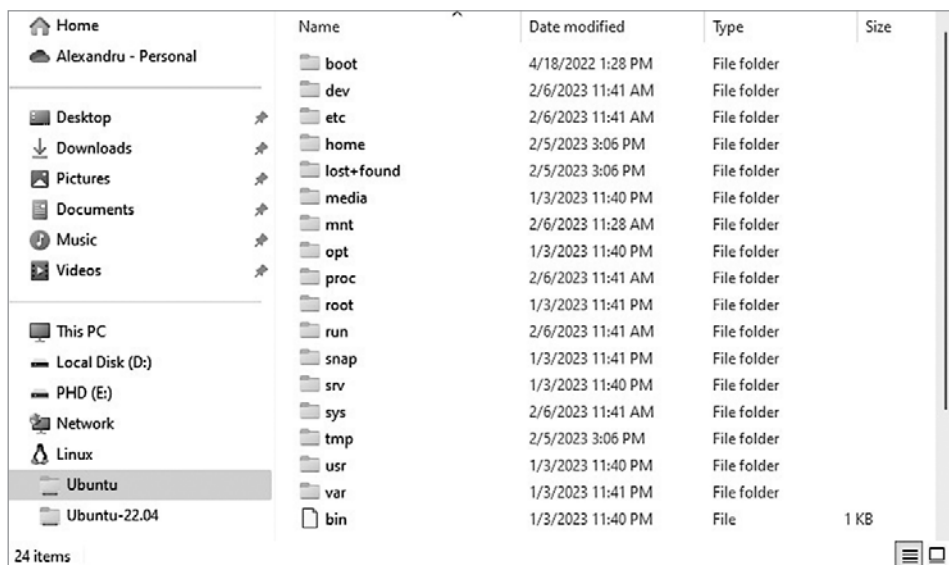


Рис. 1.11. Файловая система Ubuntu в Проводнике Windows 11

Благодаря WSL Linux быстро набирает популярность среди специалистов, которые работают под Windows. Как вы только что убедились, настроить WSL довольно легко, а для того, чтобы запустить экземпляр Linux, не нужен выделенный гипервизор.

К этому моменту вы уже знаете, как установить Linux в виртуальной машине под управлением Windows с помощью трех различных гипервизоров: VMware Workstation, Microsoft Hyper-V и Oracle VirtualBox. Как мы уже говорили, в виртуальных машинах под управлением macOS вся процедура устроена практически так же, поэтому здесь мы не будем дублировать соответствующую информацию. Интерфейс VMware Fusion в macOS почти не отличается от VMware Workstation в Windows.

Непосредственно на физическое оборудование Linux устанавливается аналогично; разница только в том, что для этого нужен физический доступ к целевым машинам. Кроме того, как мы уже говорили в начале главы, Linux можно установить еще одним, продвинутом способом — по сети. Это более сложная задача, для которой важно проявлять внимание к деталям, хотя бы на начальном уровне разбираться в том, как устроены сети, а также обязательно понимать, как происходит загрузка Linux.

В следующем разделе мы подробнее расскажем о сетевой установке.

Устанавливаем Linux: продвинутые приемы

В этом разделе мы рассмотрим продвинутые приемы установки Linux. Как мы видели в предыдущих разделах, чтобы установить ОС на физическом оборудовании или в виртуальной машине, вам понадобится физический доступ к этим машинам. Но что делать, если физического доступа нет? Или если нужно настроить так много машин, что заниматься этим вручную будет в лучшем случае утомительно, а в худшем — невыполнимо?

В корпоративной среде с десятками или сотнями компьютеров Linux можно установить с помощью автоматизированной процедуры, загружая дистрибутив по сети. Как мы уже говорили, подробный обзор механизма загрузки по сети выходит за рамки этой книги, однако мы все равно рассмотрим соответствующий процесс и разберем его важнейшие моменты, потому что эта тема в полной мере не освещена в профессиональной литературе.

Но чтобы лучше разобраться, как работает сетевая загрузка, сначала давайте вкратце рассмотрим процедуру загрузки Linux.

Процедура загрузки Linux

Как загружается Linux? Мы дадим целостное представление об этой процедуре, не вдаваясь в излишние подробности.

Когда вы впервые запускаете физическую или виртуальную машину, на которой установлен Linux, **BIOS** (или загрузочная микропрограмма) начинает загружаться и иницирует загрузчик операционной системы. BIOS содержит определенную конфигурацию и загружается производителем в микросхему ПЗУ на материнской плате (в случае физических компьютеров, а не виртуальных машин). BIOS содержит информацию об аппаратном обеспечении, о возможностях управления такими периферийными устройствами, как клавиатуры и мониторы, а также о том, какая операционная система установлена на компьютере и где находится ее загрузчик. Пользователь контролирует некоторые из этих настроек (например, последовательность загрузки или защиту паролем) и может регулировать их в соответствии со своими потребностями. BIOS также управляет **контроллерами сетевых интерфейсов (NIC)** и всеми внешними портами, включая USB и видеовыходы. Но этим возможности BIOS фактически ограничиваются: чтобы запустить любую операционную систему, которая расположена на диске, требуется ее загрузчик.

На смену BIOS пришел **UEFI** (Unified Extensible Firmware Interface, унифицированный расширяемый интерфейс встроенного ПО). Он выполняет те же функции, что и BIOS, но отличается более интерактивными интерфейсами и лучше поддерживает новые операционные системы. Впрочем, к недостаткам UEFI относится то, что его не очень хорошо поддерживают программные продукты сторонних производителей.

Стоит упомянуть также **Secure Boot** («безопасная загрузка») — функцию, которая обеспечивает дополнительный слой безопасности операционной системы и запускаемых программ. Ее поддерживают не все дистрибутивы Linux. Secure Boot использует цифровую подпись, которая подтверждает подлинность операционной системы. Чтобы операционная система поддерживала Secure Boot, ее разработчику нужно получить действительный сертификат, который можно проверить при загрузке, чтобы убедиться, что система подлинна и не подвергалась постороннему вмешательству.

Теперь, когда мы выяснили, что такое BIOS, UEFI и Secure Boot, давайте познакомимся с загрузчиком. После того как BIOS или UEFI завершит **самостоятельное при включении (POST)**, она обращается к **загрузчику операционной системы**. POST — это серия тестов, которые выполняются при запуске компьютера, чтобы убедиться, что оборудование работает исправно. Что такое загрузчик? Это связующее звено между аппаратным обеспечением и операционной системой. Он хранится в загрузочном секторе загрузочного устройства, которым может быть как раздел, так и самый первый блок носителя.

В Linux используется загрузчик **GRUB (Grand Unified Bootloader, «загрузчик Великого объединения»¹)**, который запускает ядро операционной системы. Ядро — это центральный компонент Linux, который отвечает за все программные компоненты, драйверы, службы и взаимодействие с оборудованием. Из всего перечисленного складывается так называемое пользовательское пространство. Именно GRUB способен поддерживать сетевую загрузку.

Информации из этого раздела будет достаточно, чтобы понимать, как загружается Linux. В следующем разделе мы подробно рассмотрим, как установить Linux с помощью сетевой загрузки.

Как работает сетевая загрузка с помощью PXE

Ранее в этой главе мы упоминали о том, что компьютер можно загружать с помощью **PXE**. Что же такое PXE? Это служба, которая использует различные сетевые протоколы, чтобы загружать машину по сети. PXE опирается на несколько протоколов и стандартов, которые были введены сорок лет назад, чтобы выработать крайне необходимый универсальный механизм сетевой загрузки, известный также как **NBP (Network Bootstrap Program, программа сетевой загрузки)**.

PXE основана на протоколах TFTP, DHCP, а также на стеке UDP/IP, поверх которого работает HTTP. Эти три протокола образуют платформу для интер-

¹ Игра слов: в английском языке термин *Grand Unified Theory (Теория Великого объединения)* относится к классу физических теорий, которые единым образом описывают сильное, слабое и электромагнитное взаимодействие элементарных частиц. Отдельные дистрибутивы (например, Alpine Linux) по умолчанию не используют GRUB. — Примеч. ред.

фейса прикладного программирования PXE. В большинстве сетевых карт, которые сейчас доступны на рынке, встроена микропрограмма PXE, благодаря чему PXE считается стандартным механизмом сетевой загрузки на многих архитектурах. Спецификацию последней версии PXE 2.1 можно найти по ссылке web.archive.org/web/20110524083740/http://download.intel.com/design/archives/wfm/downloads/pxespec.pdf.

Чтобы PXE работала, в сети должен быть сервер PXE, который предоставляет необходимые загрузочные файлы в ответ на запросы клиентов в сети. Для этого на сервере PXE должны быть установлены как минимум серверы DHCP и TFTP. Кроме того, необходимо также установить сервер NFS, потому что этот протокол позволяет совместно использовать файлы в сети и применяется в современных операционных системах семейства Linux.

Но прежде чем углубляться в детали, давайте обсудим, как работает сетевая загрузка. PXE опирается на клиентско-серверную среду, узлы которой оснащены сетевыми картами с поддержкой PXE. Сетевая конфигурация среды PXE была разработана так, чтобы не конфликтовать с существующей конфигурацией сети. Поскольку задействованы DHCP и TFTP, среда PXE настраивается так, чтобы не вмешиваться в существующую конфигурацию DHCP обычного маршрутизатора из локальной сети, который не относится к PXE. Это хорошо продуманная схема для корпоративной среды.

В простейшем случае, после того как все клиенты налажены так, чтобы загружаться через PXE (эта настройка доступна в BIOS почти на каждом компьютере), сетевые карты отправляют по сети запросы DHCP, чтобы обнаружить локальный сервер PXE. Чтобы правильно отвечать на эти запросы, PXE использует своего рода прокси-сервер DHCP, который отправляет клиентам, поддерживающим PXE, IP-адрес и маску сервера TFTP. В таком режиме он не вмешивается в работу сервера DHCP локальной сети.

Настройка сервера PXE выходит за рамки этой главы, но практические руководства о том, как его наладить, можно найти на страницах documentation.ubuntu.com/server/how-to/installation/how-to-netboot-the-server-installer-on-amd64 и www.redhat.com/sysadmin/pxe-boot-uefi. Впрочем, некоторые подробности, например о том, как настроить сервер DHCP, вы найдете в главе 13.

Чтобы сервер PXE работал, нужно предпринять определенные шаги в зависимости от того, какой дистрибутив вы используете. Например, можно использовать iPXE (сетевую загрузочную прошивку с открытым исходным кодом), cloud-init (характерно для Ubuntu) или kickstart (для систем семейства Fedora). Впрочем, в любом случае нужно настроить серверы DHCP, TFTP и NFS, а сервер DNS необязателен (о том, как настраивать эти серверы, вы узнаете в главе 13).

Здесь мы не будем углубляться в подробности, потому что вы еще встретитесь с ними позже. Цель этой вводной главы — познакомить вас с различными спо-

собами установки Linux, и на протяжении всей книги мы будем опираться на эти основы, чтобы постепенно подготовить вас к более сложному материалу в последующих главах.

В следующем разделе мы продемонстрируем некоторые сценарии использования разных дистрибутивов Linux в зависимости от ваших потребностей. Мы расскажем, какие дистрибутивы и приложения, по нашему мнению, лучше использовать в тех или иных практических ситуациях. Обратите внимание, что установку приложений и системы управления пакетами мы подробнее рассмотрим в главе 3.

Дистрибутивы Linux: практическое руководство

В этом разделе представлены сценарии использования, которые по большей части основаны на реальных задачах из нашего собственного опыта в области системного администрирования и разработки ПО. Каждый из этих сценариев посвящен тому, как выбрать подходящий дистрибутив Linux в той или иной ситуации.

Пример из практики: рабочая станция программиста

Этот пример опирается на сценарий, который сформулирован от лица разработчика:

Я фулстек-разработчик, пишу в основном на Java, Node.js, Python и Golang, а в качестве IDE использую прежде всего IntelliJ IDEA и VS Code. В моей среде разработки интенсивно задействованы контейнеры Docker (как для сборки, так и для развертывания), а иногда я использую виртуальные машины (на основе VirtualBox), чтобы развертывать и тестировать код локально. Мне нужна надежная и универсальная платформа для разработки.

Прежде чем решить, какой дистрибутив Linux подходит для этих задач, давайте рассмотрим функциональные и системные требования:

- **Функциональные требования.** Задачи подразумевают относительно мощную платформу для повседневной разработки — либо настольный компьютер, либо ноутбук. Чтобы развертывать и тестировать код, разработчик пользуется локальными ресурсами (например, контейнерами Docker и виртуальными машинами), возможно, часто в автономном режиме (например, в самолете или еще где-то в дороге).
- **Системные требования.** Система будет в основном использовать среду рабочего стола Linux и диспетчер окон, где контекст часто переключается между IDE и окнами терминала. Должны быть легко доступны необходимые программные пакеты для IDE, Docker, гипервизора (VirtualBox) и других инструментов разработки (программ с открытым исходным кодом или ком-

мерческих решений); в идеале все это программное обеспечение должно всегда быть свежим, причем устанавливаться и настраиваться с минимальными усилиями.

Какой дистрибутив Linux выбрать

Из дистрибутивов лучше всего выбрать **Ubuntu Desktop LTS**, который относительно стабилен, работает практически на любой аппаратной платформе и по большей части обеспечивает актуальные драйверы для оборудования. Для необходимых приложений и инструментов, как правило, доступны стабильные программные пакеты, которые часто обновляются. Ubuntu LTS — это экономичная и безопасная операционная система корпоративного уровня, которая подходит как для организаций, так и для домашних пользователей.

Помимо Ubuntu, для рабочей станции программиста одинаково хорошо подходят **Fedora** и **openSUSE**. Выбирать из них имеет смысл в зависимости от того, какая экосистема вам нужна — на основе **Debian** или на основе **Red Hat/SUSE**, а также от того, важны ли вам более актуальные пакеты.

Пример из практики: защищенный веб-сервер

Этот пример опирается на сценарий, который сформулирован от лица инженера DevOps:

Я ищу надежную платформу, на которой будет работать безопасный и относительно легкий веб-сервер корпоративного класса. Он обрабатывает запросы HTTP/SSL, разгружая SSL перед тем, как маршрутизировать запросы к другим внутренним веб-серверам, сайтам и конечным точкам API. Балансировать нагрузку не требуется.

В этом примере давайте рассмотрим **функциональные требования**. Когда идет речь о безопасных веб-серверах корпоративного класса с открытым исходным кодом, в качестве основных вариантов обычно рассматривают nginx, Apache HTTP Server, Node.js, Apache Tomcat и lighttpd. Поскольку сейчас речь не о том, как выбрать веб-сервер, мы предположим, что предпочли Apache HTTP Server, который поддерживает современные технологии SSL/TLS, отличается высокой производительностью и относительно просто настраивается.

Этот сервер можно развернуть в среде VPS, в локальном центре обработки данных или в публичном облаке, в форм-факторе виртуальной машины или контейнера Docker. Нам нужна относительно компактная платформа Linux корпоративного класса.

Какой дистрибутив Linux выбрать

Здесь лучше рассмотреть **Rocky Linux** или **AlmaLinux**: в большинстве случаев эти два дистрибутива идеально подходят для Apache HTTP Server. Они отно-

сительно легковесны и включают в себя только базовые серверные компоненты и сетевой стек операционной системы. И Rocky, и Alma широко доступны в виде шаблонов для развертывания VPS как у частных, так и у публичных облачных провайдеров. Наш Apache HTTP Server можно запустить в контейнере Docker поверх Rocky Linux или AlmaLinux, потому что нам может понадобиться горизонтально масштабировать сервер, развертывая несколько его экземпляров. В главе 13 мы подробнее поговорим о том, как настраивать веб-сервер.

Пример из практики: личный блог

Этот пример опирается на сценарий, который сформулирован от лица программиста-блогера:

Я хочу завести блог о разработке программного обеспечения. Буду использовать для блога движок Ghost, который работает на Node.js, а в качестве серверной базы данных — MySQL. Блог будет размещен на VPS одного из крупных облачных провайдеров. Я собираюсь сам устанавливать, поддерживать и сопровождать всю систему. Какой дистрибутив Linux лучше выбрать?

Давайте обсудим **функциональные требования** для этого сценария. Мы ищем решение для самоуправляемого публичного VPS, а значит, стоимость соответствующего хостинга будет важным фактором. Также требуется, чтобы обслуживать необходимые программные пакеты было относительно просто. Мы ориентируемся на то, что программное обеспечение (включая саму платформу Linux) будет часто обновляться.

Какой дистрибутив Linux выбрать

В качестве дистрибутива Linux для этого сценария мы бы выбрали **Debian Stable** или **Ubuntu Server LTS**. Как уже отмечалось, Ubuntu — это надежный и безопасный дистрибутив корпоративного класса. **Debian** не менее стабилен и тоже хорошо подходит для целевых приложений. Оба дистрибутива можно обслуживать и администрировать без больших усилий, а нужные программные пакеты — Node.js, Ghost и MySQL — легкодоступны и хорошо поддерживаются. Ubuntu Server занимает относительно мало места; в рамках его системных требований можно легко развернуть необходимый стек программного обеспечения для блога, поэтому затраты на хостинг будут разумными.

Пример из практики: медиасервер

Этот пример опирается на сценарий, который сформулирован от лица пользователя домашнего кинотеатра:

У меня довольно большая коллекция фильмов (резервные копии дисков DVD/Blu-ray), видео, фотографий и других медиафайлов, которые хранятся в сетевом хранилище (NAS). У него есть встроенный медиасервер, но произво-

длительность потоковой передачи довольно низкая. В качестве проигрывателя я использую Plex, а в качестве медиасервера — Plex Media Server. Какой дистрибутив Linux мне лучше выбрать?

Исходя из этого описания, давайте определим **системные требования** для такого сценария. Важнейшие из них — скорость (чтобы плавно передавать видео в высоком разрешении), безопасность и стабильность. Соответствующие программные пакеты и медиакодеки часто обновляются, поэтому задачи по обслуживанию и модернизации платформы тоже возникают довольно часто. Платформа размещена локально, на мощном настольном компьютере с большим объемом памяти. Медиафайлы транслируются из сетевого хранилища по локальной сети, где данные доступны по протоколу NFS.

Какой дистрибутив Linux выбрать

Для медиасервера отлично подойдет как **Debian**, так и **Ubuntu**. В сообществе Linux стабильный выпуск Debian считается чрезвычайно надежным, хотя и немного старомодным. У обеих систем развитые сетевые возможности и механизмы безопасности, но решающим фактором при выборе между ними может стать то, что у Plex Media Server есть пакет для Debian, совместимый с архитектурой Arm, а пакет медиасервера для Ubuntu доступен только для платформ Intel/AMD. Если бы мы ориентировались на малогабаритное устройство с процессором Arm, то стоило бы выбрать Debian; в противном случае **Ubuntu LTS** тоже вполне подойдет.

Теперь, когда мы рассмотрели различные сценарии использования, настало время выбрать дистрибутив Linux и начать с ним практиковаться. Надеемся, что внушительный объем информации, которую мы предоставили в этой главе, окажется бесценным, когда вы начнете осваивать Linux на практике.

Итоги

В этой главе мы познакомились с дистрибутивами Linux, уделив особое внимание тому, как выбрать подходящую платформу для тех или иных задач и выполнить соответствующие процедуры установки.

На протяжении всей главы мы делали основной акцент на дистрибутиве Ubuntu, а также в духе практического подхода рассказали о том, как Linux работает в виртуальных машинах. Мы также сделали небольшой экскурс в мир Windows, где рассмотрели WSL — современную абстракцию Linux в форме «родного» приложения для Windows.

Мы надеемся, что в этой главе вы овладели навыками, которые помогут эффективно выбирать те или иные дистрибутивы Linux в зависимости от своих потребностей. Вы узнали, как устанавливать и настраивать Linux на различных платформах. Кое-что из этого пригодится вам на протяжении всей книги, но

еще важнее, что теперь вы сможете быстро установить и опробовать нужный дистрибутив Linux.

Далее мы начнем подробнее рассматривать различные подсистемы, компоненты, службы и приложения Linux. В главе 2 «Командная оболочка и файловая система Linux» вы узнаете, как устроена файловая система Linux, и познакомитесь с соответствующими инструментами.

Вопросы

Вот несколько вопросов и тем для размышления. Одни из них опираются на материал этой главы, а другие относятся к последующим частям книги:

1. Если у нас довольно много экземпляров или дистрибутивов Linux, которые развернуты и работают под управлением виртуальных машин, то как упростить управление ими?

Подсказка: используйте **Vagrant** — инструмент, который позволяет создавать среды виртуальных машин и управлять ими.

2. Можно ли запустить несколько экземпляров Linux под управлением WSL?

Подсказка: да, можно.

Дополнительная литература

Вот книги издательства Packt, которые помогут вам справиться с установкой Linux:

- Оливер Пельц (Oliver Pelz). «Fundamentals of Linux».
- Джей Лакруа (Jay LaCroix). «Mastering Ubuntu Server».

2

Командная оболочка и файловая система Linux

Современному специалисту по Linux важно разбираться в том, как устроена **файловая система Linux** и как **управлять файлами**, а также владеть основами **командной оболочки Linux** и **интерфейсом командной строки (CLI)**.

В этой главе вы научитесь использовать командную оболочку Linux и некоторые распространенные команды в ней. Вы узнаете, как устроены простые команды и файловая система Linux. Мы рассмотрим различные команды для работы с файлами и каталогами, а попутно познакомимся с наиболее распространенными текстовыми редакторами с интерфейсом командной строки. Надеемся, что к концу этой главы вы начнете уверенно пользоваться командной строкой Linux и будете готовы к более продвинутым темам. Эта глава заложит основу для работы с оболочкой Linux, а подробнее мы поговорим об оболочке в главе 8 «Сценарии командной оболочки в Linux».

Мы рассмотрим такие темы:

- Знакомство с командной оболочкой Linux.
- Файловая система Linux.
- Как работать с файлами и каталогами.
- Как создавать и редактировать файлы с помощью текстовых редакторов.

Технические требования

Для примеров из этой главы требуется действующий экземпляр стандартного дистрибутива Linux, который установлен на сервере, настольном компьютере (или ноутбуке) либо виртуальной машине. В наших примерах используются Ubuntu и Fedora, однако все рассмотренные команды и упражнения аналогично ведут себя в любых других дистрибутивах Linux.

Первое знакомство с оболочкой Linux

Linux происходит от операционной системы Unix, одним из главных достоинств которой был интерфейс командной строки — он также известен как *командная*

оболочка или *командный интерпретатор*. В Unix оболочка вызывается командой `sh`. Оболочка — это программа, у которой есть два потока — *входной* и *выходной*. Входной поток — команда, которую ввел пользователь, а выходной — результат того, как компьютер выполнил или интерпретировал эту команду. Другими словами, командная оболочка — основной интерфейс, с помощью которого пользователь взаимодействует с машиной.

Самая популярная оболочка в распространенных дистрибутивах Linux называется **Bash** — это аббревиатура от **Bourne Again Shell**¹; такое название дано в честь Стива Борна, который создал классическую командную оболочку в Unix. Помимо Bash, в Linux существуют и другие оболочки, такие как **ksh**, **tcsch** и **zsh**. В этой главе, как и на протяжении всей книги, мы будем работать в оболочке Bash, потому что она особенно широко используется в современных дистрибутивах Linux.

Во многих дистрибутивах — например, Debian, Ubuntu, Fedora, CentOS Stream, RHEL, openSUSE, SLE и Linux Mint — по умолчанию используется командная оболочка *Bash*, а другие дистрибутивы — скажем, Kali Linux и некоторые издания Manjaro — перешли на *zsh* по умолчанию. Если вы пользуетесь macOS, то, скорее всего, знаете, что в этой системе *zsh* уже несколько лет служит стандартной оболочкой. Впрочем, в Linux можно установить любую оболочку и назначить ее оболочкой по умолчанию. В целом все оболочки очень похожи друг на друга, потому что делают одно и то же, однако они различаются *дополнениями* по части функций и удобства использования. Если вас интересует какая-то конкретная оболочка, практикуйтесь в ней и изучайте различия между другими оболочками.

Каждому пользователю можно назначить одну оболочку, причем у разных пользователей одной системы могут быть разные оболочки. Чтобы узнать, какая оболочка связана с каждым пользователем по умолчанию, можно, например, обратиться к файлу `/etc/passwd`. Подробнее об этом файле и об учетных записях пользователей мы поговорим в главе 4 «Как управлять пользователями и группами», а пока просто важно знать, где искать оболочку по умолчанию: в каждой строке файла `/etc/passwd` ей соответствует путь после последнего двоеточия (:). В строках перечислены сведения о пользователях, в том числе идентификаторы их пользователей (**UID**) и групп (**GID**), имена пользователей, их домашние каталоги и соответствующие оболочки.

Чтобы узнать, какая оболочка назначена тому или иному пользователю, запустите такую команду, указав имя пользователя (в нашем случае `packt`):

```
cat /etc/passwd | grep packt
```

¹ Игра слов: Bourne Again буквально означает «снова Борн», но при этом созвучно фразе *born again* («заново рожденный»). — *Примеч. ред.*

В результате вы увидите строки файла `/etc/passwd`, которые содержат `packt`. А чтобы узнать *текущую оболочку*, можно поступить проще: запустить такую команду:

```
echo $0
```

Эта команда выводит сведения о том, какая именно программа ее выполнила: если вы запускаете команду в командной строке, то выполняющая программа — это и есть оболочка. Часть `$0` — это **специальный параметр Bash**, который ссылается на текущий запущенный процесс. На следующем снимке экрана показано, что выводят две предыдущие команды, с помощью которых мы узнавали имя оболочки, а также чем отличается вывод команды `echo $0` в Ubuntu от Debian:



```
packt@neptune:~$ cat /etc/passwd | grep packt
packt:x:1000:1000:alexandru calcatinge:/home/packt:/bin/bash
packt@neptune:~$ echo $0
-bash

alexandru@debian: ~
alexandru@debian:~$ echo $0
/bin/bash
alexandru@debian:~$
```

Рис. 2.1. Команды, которые позволяют узнать имя оболочки

Как видите, в разных дистрибутивах команда `echo $0` приводит к разным результатам, но с одной и той же информацией: запущена оболочка Bash. Если вы предпочитаете другую оболочку, которая уже установлена в системе, вы легко можете назначить ее своему пользователю. Однако если вы знаете Bash, то сможете без труда работать и с другими доступными оболочками.

Командные оболочки Linux чувствительны к регистру. Не забывайте об этом, когда вводите что-то в командной строке. Например, команда `cat`, которую мы недавно использовали, пишется строчными буквами. Если вы введете `Cat` или `CAT`, оболочка не распознает это как команду. Тот же принцип касается путей к файлам. Обратите внимание, что имена стандартных каталогов в вашем домашнем каталоге начинаются с прописной буквы: `~/Documents`, `~/Downloads` и т. д. Это не то же самое, что имена `~/documents` или `~/downloads`.

В этой главе вы освоите команды и оболочку Linux, а также познакомитесь с файловой системой Linux. О том, как управлять программным обеспечением, пойдет речь в главе 3, поэтому если бы мы сейчас продемонстрировали, как установить другую оболочку, то опередили бы события. Нам хочется, чтобы вы неторопливо, но уверенно наращивали свои знания о Linux, поэтому устанав-

мы остановимся на Bash, которую будем использовать на протяжении всей книги.

Чтобы узнать, какие оболочки установлены в вашей системе, запустите такую команду:

```
cat /etc/shells
```

На следующей иллюстрации показано, что выводит эта команда в нашем случае: список всех оболочек, которые по умолчанию установлены в нашем Ubuntu Server 22.04.2 LTS:

```
packt@neptune:~$ cat /etc/shells
# /etc/shells: valid login shells
/bin/sh
/bin/bash
/usr/bin/bash
/bin/rbash
/usr/bin/rbash
/usr/bin/sh
/bin/dash
/usr/bin/dash
/usr/bin/tmux
/usr/bin/screen
```

Узнав, какие оболочки есть в системе, вы можете использовать любую из них или установить новые, как мы покажем в главе 3. Кроме того, в главе 4, где мы будем работать с учетными записями пользователей, вы узнаете, как сменить оболочку для того или иного пользователя.

В следующем разделе поговорим о том, как подключиться к командной оболочке.

Как подключиться к командной оболочке

К командной оболочке можно подключиться двумя способами: **tty** и **pts**. Название **tty** — сокращение от **teletypewriter** (телетайп); так назывался тип терминала, который использовался на заре вычислительной техники. Это непосредственное соединение, при котором порты напрямую подключены к компьютеру. Пользователь может вводить данные прежде всего с помощью клавиатуры, которая считается стандартным терминальным устройством.

Подключение **pts** создается в соединениях по SSH или Telnet. Его название расшифровывается как **pseudo-terminal slave** (ведомое устройство псевдотерминала) и это эмулированное (опосредованное) соединение; его чаще всего создают такие программы, как **ssh** или **xterm**. Это подчиненная часть **псевдо-терминала**, который обозначается **pty**.

В следующем разделе мы подробнее рассмотрим, как подключаться к виртуальным терминалам, которые доступны в Linux.

Виртуальные консоли/терминалы

Под **терминалом** традиционно понималось устройство, которое передает строки ввода (то есть команды) между процессом и другими устройствами ввода-вывода, такими как клавиатура и экран. Существуют также **псевдотерминалы** — это виртуальные терминалы, которые ведут себя так же, как **классический терминал**, однако не взаимодействуют с устройствами напрямую, а эмулируют ядром Linux, которое передает ввод-вывод командной оболочке.

Виртуальные консоли работают в фоновом режиме, даже если соответствующие терминалы не открыты. Для доступа к ним можно использовать сочетания клавиш от **Ctrl + Alt + F1** до **Ctrl + Alt + F6**, которые открывают на компьютере консоли **tty1** – **tty6** соответственно.

Мы продемонстрируем это на примере Ubuntu 22.04.2 LTS Server, которая функционирует в виртуальной машине, но в Rocky Linux все работает так же. Если запустить виртуальную машину и дожидаться запроса на вход в систему, то первая строка на экране будет выглядеть примерно так (**neptune** — это имя, которое мы дали виртуальной машине):

```
Ubuntu 22.04.2 LTS neptune tty1
```

Если вы нажмете любое из перечисленных сочетаний клавиш, то увидите, как терминал переключится с **tty1** на другой экземпляр **tty**. Например, если нажать **Ctrl + Alt + F6**, вы увидите следующее:

```
Ubuntu 22.04.2 LTS neptune tty6
```

Здесь мы использовали серверную версию Ubuntu, в которой не установлен графический интерфейс. Но если вы работаете с настольной версией, то с помощью **Ctrl + Alt + F7** можете переключиться в графический режим (например, диспетчер окон X).

На случай, когда нет возможности использовать эти комбинации клавиш, существует специальная команда **chvt**, которая переключает виртуальный терминал и запускается в формате **chvt номер_терминала** (например, **chvt 2**). Хотя мы еще не обсуждали команды оболочки, мы покажем, как использовать некоторые из них. Дальнейшие действия можно выполнить только под учетной записью администратора или с помощью команды **sudo**. Вкратце, **sudo** расшифровывается как *superuser do*¹ и позволяет любому пользователю запускать программы с правами администратора или другого пользователя. (Подробнее об этом см. в главе 4 «Как управлять пользователями и группами».)

Далее мы на примере Ubuntu продемонстрируем, как переключать виртуальные терминалы. Сначала посмотрим, какой терминал мы используем в данный момент, а затем сменим его на другой, не применяя клавиши **Ctrl + Alt + Fn**.

¹ Сделать как суперпользователь. — *Примеч. пер.*

Команда `who` показывает, какие пользователи сейчас аутентифицированы в системе компьютера. Поскольку мы подключены к виртуальной машине по SSH, в нашем случае команда покажет, что пользователь `packt` использует нулевой псевдотерминал (`pts/0`):

```
packt      pts/0          2023-02-28 10:45 (192.168.122.1)
```

Если бы мы выполнили ту же команду непосредственно в консоли виртуальной машины, то получили бы такой результат:

```
packt      pts/0          2023-02-28 10:45 (192.168.122.1)
packt      tty1          2023-02-28 10:50
```

Здесь видно, что пользователь подключен как к виртуальному терминалу 1 (`tty1`), так и к псевдотерминалу (`pts/0`) по SSH из управляющей операционной системы.

Теперь посмотрим, как с помощью `chvt` переключиться на виртуальный терминал 6. Если запустить команду `sudo chvt 6`, система предложит ввести пароль, после чего вы сразу же переключитесь на шестой терминал. Если снова запустить `who`, вы увидите список всех пользователей, которые вошли в систему, и их виртуальных терминалов — в нашем случае это `pts/0`, `tty2` и `tty6`. Имейте в виду, что в вашей системе вывод может отличаться — например, в нем могут быть другие номера терминалов.

Теперь, когда мы знаем, какими способами можно подключаться к командной оболочке, давайте познакомимся с приглашением командной строки.

Приглашение командной строки

Приглашение командной строки (или **приглашение оболочки**) отображается в начале строки, в которой вы вводите команды. Обычно приглашение содержит имя пользователя, имя узла, текущий рабочий каталог и символ, который указывает на тип текущего пользователя.

Вот как выглядит приглашение командной строки в серверной версии Ubuntu 22.04.2 LTS (аналогичная картина будет и в Debian):

```
packt@saturn:~$
```

А вот пример из серверной версии Fedora 37 (такое же приглашение вы увидите в Rocky Linux, RHEL или AlmaLinux):

```
[packt@localhost ~]$
```

Вот что означают части этого приглашения:

- `packt` — имя пользователя, который вошел в систему и работает в ней в данный момент.
- `saturn` и `localhost` — имена узлов (компьютеров).

- Знак `~` (тильда) обозначает домашний каталог.
- Знак `$` (доллар) обозначает, что вы — обычный пользователь. Если вы вошли в систему как администратор, знак меняется на `#` (решетку).

В openSUSE приглашение командной строки отличается от Ubuntu/Debian и Fedora/RHEL. Вот как оно выглядит в серверной версии Leap 15.4:

```
packt@localhost:~>
```

Как видите, здесь нет ни доллара, ни решетки, но есть знак `>` (больше). Поначалу это может сбить с толку, но если вы переключитесь в учетную запись пользователя `root`, то знак сменится на решетку, например:

```
localhost:/home/packt #
```

Далее рассмотрим, какие типы команд бывают в командной оболочке.

Типы команд оболочки

Оболочки работают с **командами**, которые бывают двух типов: внутренние и внешние. **Внутренние команды** встроены в оболочку, а **внешние** устанавливаются отдельно. Чтобы узнать, к какому типу относится та или иная команда, используйте команду `type`. Например, так можно проверить тип команды `cd`:

```
packt@neptune:~$ type cd
cd is a shell builtin1
```

Как видите, `cd` — это внутренняя команда, которая встроена в оболочку. Если вас интересуют типы других команд, о которых вы узнаете в следующих разделах, аналогично введите `type` перед именем команды. Рассмотрим еще несколько примеров:

```
packt@neptune:~$ type date
date is /usr/bin/date2
```

```
packt@neptune:~$ type ls
ls is aliased to `ls --color=auto`3
```

```
packt@neptune:~$ type man
man is /usr/bin/man
```

```
packt@neptune:~$ type grep
grep is aliased to `grep --color=auto`
```

¹ `cd` — встроенная команда оболочки. — *Примеч. пер.*

² `date` — внешняя команда, расположенная в `/usr/bin/date`. — *Примеч. пер.*

³ `ls` — псевдоним команды `ls --color=auto`. Чтобы увидеть расположение команды, введите `type -a ls`. — *Примеч. пер.*

```
packt@neptune:~$ type echo
echo is a shell builtin
```

```
packt@neptune:~$ type touch
touch is /usr/bin/touch
```

```
packt@neptune:~$ type pwd
pwd is a shell builtin
```

```
packt@neptune:~$ type elif
elif is a shell keyword1
```

Теперь, когда вы знаете некоторые типы команд Linux, давайте разберемся, как устроены команды и из каких компонентов они состоят.

Как устроены команды Linux

Мы уже использовали некоторые команды, но не говорили о том, как вообще устроены команды в Linux. Сейчас мы исправим это упущение, чтобы вам стало понятнее, как обращаться с командами. В простейшей форме команды Unix и Linux состоят из таких компонентов:²

- Имя команды
- Ключи (флаги) команды
- Аргументы команды

В оболочке команды используются в такой общей форме:

```
имя_команды [-ключи] [аргументы]
```

В качестве примера рассмотрим команду `ls` — одну из самых популярных в Linux. Название команды происходит от *list* (*перечислять*); она выводит список файлов и каталогов, и ее можно запускать как с ключами, так и с аргументами.

Если запустить `ls` в простейшей форме, без ключей и аргументов, она выводит список содержимого вашего текущего рабочего каталога (его можно узнать командой `pwd`). В нашем случае это домашний каталог, на который указывает символ `~` в приглашении оболочки.

¹ *elif* — ключевое слово командной оболочки. — *Примеч. пер.*

² В терминологии POSIX любая отдельная часть команды называется аргументом (argument). Первый аргумент — это имя утилиты (utility name). За ним идут параметры, или опции (options; например, `-a`), причем у параметров могут быть аргументы параметров (option arguments). Последняя часть команды — операнды (operands); в книге именно они называются аргументами. Чтобы уменьшить путаницу, в этом переводе части команды называются именем, ключами и аргументами. — *Примеч. пер.*

Команда `ls` с ключом `-l` (строчная буква `L`) выводит список в расширенном формате, предоставляя дополнительную информацию о файлах и каталогах из текущего каталога:

```
packt@saturn:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  snap  Templates  Videos

packt@saturn:~$ ls -l
total 36
drwxr-xr-x 2 packt packt 4096 Feb 28 12:32 Desktop
drwxr-xr-x 2 packt packt 4096 Mar  1 21:05 Documents
drwxr-xr-x 2 packt packt 4096 Feb 28 12:32 Downloads
drwxr-xr-x 2 packt packt 4096 Feb 28 12:32 Music
drwxr-xr-x 2 packt packt 4096 Feb 28 12:32 Pictures
drwxr-xr-x 2 packt packt 4096 Feb 28 12:32 Public
drwx----- 3 packt packt 4096 Feb 28 12:32 snap
drwxr-xr-x 2 packt packt 4096 Feb 28 12:32 Templates
drwxr-xr-x 2 packt packt 4096 Feb 28 12:32 Videos

packt@saturn:~$ ls -l ~/Documents
total 0
-rw-rw-r-- 1 packt packt 0 Mar  1 21:04 file1
-rw-rw-r-- 1 packt packt 0 Mar  1 21:04 file2
-rw-rw-r-- 1 packt packt 0 Mar  1 21:04 file3
-rw-rw-r-- 1 packt packt 0 Mar  1 21:05 report1
-rw-rw-r-- 1 packt packt 0 Mar  1 21:05 report2
-rw-rw-r-- 1 packt packt 0 Mar  1 21:05 report3
-rw-rw-r-- 1 packt packt 0 Mar  1 21:05 report4
-rw-rw-r-- 1 packt packt 0 Mar  1 21:05 reports
```

В предыдущем примере мы запустили команду `ls -l ~/Documents/`, чтобы вывести содержимое каталога `~/Documents`. Здесь показано, как использовать команду с ключами и аргументами, не меняя текущий рабочий каталог на `~/Documents`.

В следующем разделе вы узнаете, как использовать справочные страницы, которые доступны по умолчанию в Linux.

Справочные страницы Linux

Лучший друг системного администратора Linux — справочные страницы (ман-страницы). У каждой команды в Linux есть справочная страница, которая предоставляет подробную информацию об ее использовании, ключах и аргументах¹. Чтобы открыть справочную страницу той или иной команды, запусти-

¹ Справочные страницы в Linux по умолчанию написаны на английском языке. С самого начала существования Unix (и впоследствии также Linux) предпринималось много инициатив по переводу справочных страниц на русский язык, но на сегодняшний день ни одна из них не стала хоть в какой-то мере общепринятой в среде русскоязычных специалистов по Linux. — *Примеч. пер.*

те `man имя_команды`. Например, справку по команде `ls` можно вызвать с помощью `man ls`.

Справочная страница каждой команды разбита на несколько разделов, причем названия разделов и их структура более-менее унифицированы во всех дистрибутивах. Основные разделы таковы: `name` (имя), `synopsis` (обзор), `configuration` (конфигурация), `description` (описание), `options` (ключи/флаги), `exit status` (код выхода), `return value` (возвращаемое значение), `errors` (ошибки), `environment` (окружение), `files` (файлы), `versions` (версии), `conforming to` (соответствие стандартам), `notes` (примечания), `bugs` (дефекты), `examples` (примеры), `authors` (авторы), `copyright` (сведения об авторских правах) и `see also` (см. также).

Помимо справочных страниц, почти у всех команд в Linux есть ключ `--help`, который позволяет получить обзорную справку.

Чтобы подробнее узнать о том, как использовать команды `help` и `man`, можно вызвать их собственную справку. Попробуйте запустить такие команды:

```
$ man man
$ help help
```

Имейте в виду, что справочные страницы — это не пошаговое руководство для начинающих, а профессиональная техническая документация, которая поначалу может показаться сложной. Мы советуем обращаться к `man`-страницам как можно чаще: прежде чем искать что-то в интернете, попробуйте сначала прочитать справку. Эти полезные упражнения помогут вам быстрее освоить команды Linux.

Считайте справочные страницы своей «палочкой-выручалочкой», подобно учебникам, которыми вы пользовались в школе. На этих страницах вы найдете информацию из первых рук именно тогда, когда она понадобится. А если учесть ситуации, когда доступ в интернет ограничен, а поисковые системы недоступны, то вы еще больше оцените встроенные справочные страницы. Научитесь обращаться их себе на пользу.

В следующем разделе вы узнаете о файловой системе Linux.

Файловая система Linux

Файловая система Linux представляет собой логическую структуру файлов, которые хранятся в разделе или на диске. На жестком диске может быть один или несколько разделов, а в каждом разделе чаще всего содержится только одна файловая система, причем она может, в частности, занимать весь диск. Например, одна файловая система может быть корневой (она обозначается символом `/`), а другая — `/home`. Но может быть и один раздел, который содержит в себе всю систему.

Как правило, считается хорошей практикой размещать по одной файловой системе в каждом разделе, потому что такую структуру логически легче обслуживать и сопровождать. В Linux все является файлами, так что физические устройства — такие, как жесткие диски, приводы DVD, устройства USB и дисководы, — тоже рассматриваются как файлы. В этом разделе вы узнаете о структуре каталогов и о том, как работать с файлами, а также овладеете некоторыми весьма полезными приемами для того, чтобы редактировать файлы из командной строки.

Структура каталогов

В Linux файловая система организована в виде **иерархической** структуры. Она похожа на перевернутое дерево, которое начинается с корневого каталога (/), а от него расходятся все остальные ветви (каталоги) файловой системы.

Структуру файловых систем в системах семейства Unix определяет стандарт **FHS (Filesystem Hierarchy Standard)**, стандарт иерархии файловой системы), хотя в файловых системах Linux также содержатся отдельные каталоги, которые еще не регламентированы стандартом.

Исследуем файловую систему Linux из командной строки

Попробуйте самостоятельно исследовать файловую систему с помощью команды **tree**. В Fedora Linux она уже установлена, но если вы используете Ubuntu, вам понадобится установить ее с помощью такой команды:

```
$ sudo apt install tree
```

Не бойтесь изучать файловую систему: ведь ничего не сломается от того, что вы просто осматриваетесь в ней. Чтобы просматривать содержимое каталогов, можно применять команду **ls**, но **tree** выводит их иерархию в наглядном виде. В следующем примере показано, чем различается вывод этих команд:

```
packt@neptune:~$ ls -la
total 36
drwxr-x--- 4 packt packt 4096 Feb 28 10:37 .
drwxr-xr-x 3 root  root  4096 Feb 27 08:58 ..
-rw----- 1 packt packt 1039 Mar  1 18:56 .bash_history
-rw-r--r-- 1 packt packt  220 Jan  6 2022 .bash_logout
-rw-r--r-- 1 packt packt 3771 Jan  6 2022 .bashrc
drwx----- 2 packt packt 4096 Feb 27 08:59 .cache
-rw----- 1 packt packt   0 Feb 27 10:52 .lessht
-rw-r--r-- 1 packt packt  807 Jan  6 2022 .profile
drwx----- 2 packt packt 4096 Feb 27 08:58 .ssh
-rw-r--r-- 1 packt packt   0 Feb 27 08:59 .sudo_as_admin_successful
-rw-rw-r-- 1 packt packt   67 Feb 27 14:33 users
```

```
packt@neptune:~$ tree -a
.
├── .bash_history
```

```

├── .bash_logout
├── .bashrc
├── .cache
│   └── motd.legal-displayed
├── .lessht
├── .profile
├── .ssh
│   └── authorized_keys
├── .sudo_as_admin_successful
├── users
2 directories, 9 files

```

У команды **tree** много ключей, о которых можно узнать из справочной страницы. Давайте запустим **tree** с ключом **-L**, который указывает, на сколько уровней спускаться, а последним аргументом будет каталог, с которого нужно начать. В нашем примере команда спускается на один уровень, начиная с корневого каталога, который обозначен косой чертой:

```
$ tree -L 1 /
```

Давайте изучим каталоги, список которых вывела команда **tree**:

```

packt@neptune:~$ tree -L 1 /
/
├── bin -> usr/bin
├── boot
├── dev
├── etc
├── home
├── lib -> usr/lib
├── lib32 -> usr/lib32
├── lib64 -> usr/lib64
├── libx32 -> usr/libx32
├── lost+found
├── media
├── mnt
├── opt
├── proc
├── root
├── run
├── sbin -> usr/sbin
├── snap
├── srv
├── swap.img
├── sys
├── tmp
├── usr
└── var

```

Учтите, что некоторые каталоги, к которым вы будете обращаться, содержат очень много файлов и/или других каталогов, и их список может загромоздить окно терминала.

Перечислим каталоги, которые есть почти во всех версиях Linux. Вот краткий обзор основных каталогов файловой системы Linux:

- `/`: корневой каталог, в который вложены все остальные каталоги.
- `/bin`: двоичные файлы основных команд. Здесь хранятся программы в двоичном формате.
- `/boot`: статические файлы загрузчика. Здесь хранится ядро, а также загрузчик и `initramfs`.
- `/dev`: файлы устройств. Узлы для аппаратных и программных устройств, список устройств ядра.
- `/etc`: конфигурация системы, специфичная для данного компьютера. Основные файлы конфигурации, загрузочные сценарии, `crontab`, таблицы дисковых устройств `fstab`, файл учетных записей `passwd`.
- `/home`: домашний каталог пользователя. Здесь хранятся пользовательские файлы.
- `/lib`: основные общие библиотеки и модули ядра. Общие библиотеки похожи на файлы **DLL** в Windows.
- `/media`: точка монтирования для съемных носителей. Предназначена для периферийных устройств и внешних носителей USB.
- `/mnt`: точка временного монтирования файловой системы. Используется в устаревших системах.
- `/opt`: пакеты вспомогательных прикладных программ. Сюда устанавливается *необязательное* программное обеспечение.
- `/proc`: виртуальная файловая система под управлением ядра. Эта специальная структура каталогов содержит файлы, без которых система не сможет нормально функционировать.
- `/sbin`: основные системные двоичные файлы, которые тоже жизненно важны для работы системы.
- `/srv`: сведения о службах, которые предоставляет система.
- `/tmp`: временные файлы¹.
- `/usr`: иерархия второго уровня. Это самый большой каталог в Linux, и в нем находятся файлы, которые нужны, чтобы обслуживать обычных пользователей системы:
 - `/usr/bin`: исполняемые файлы программ;
 - `/usr/lib`: общие библиотеки для программ из `/usr/bin`;
 - `/usr/local`: программы, которые скомпилированы из исходного кода и не включены в дистрибутив;

¹ Помимо каталога `/tmp` в системах встречается каталог `/var/tmp`. Предназначение первого — хранить временные данные до первой перезагрузки, второго — сохранять временные данные между перезагрузками. — *Примеч. ред.*

- `/usr/sbin`: специальные программы для системного администрирования;
- `/usr/share`: данные, которыми совместно пользуются программы в `/usr/bin`: такие, как файлы конфигурации, значки, обои или звуковые файлы;
- `/usr/share/doc`: файлы документации для любых программ и других ресурсов.
- `/var`: изменяемые данные. Здесь хранятся только такие данные, которые пользователь может модифицировать: базы данных, файлы очереди печати, электронная почта и др.
- `/var/log`: содержит файлы журнала, которые протоколируют деятельность системы.

Далее мы узнаем, как работать с этими файлами и каталогами.

Как работать с файлами и каталогами

Помните, что в Linux все является файлом¹, и каталог — это тоже файл. Поэтому очень важно уметь обращаться с файлами, а для этого понадобится освоить команды, которые позволяют выполнять основные операции с файлами и каталогами — в том числе создавать и просматривать файлы, управлять их расположением, редактировать свойства файлов и создавать ссылки.

Пути к файлам

У каждого файла в файловой системе есть *путь* — то есть местоположение файла, которое представлено в удобочитаемом виде. В Linux все файлы содержатся в корневом каталоге и структурированы в соответствии со стандартом FHS, а иерархические отношения между файлами и каталогами выражаются с помощью косой черты (`/`). В вычислительной технике этот символ исторически разделял части адресов. Пути — это, по сути, адреса файлов.

В Linux бывают пути двух типов: относительные и абсолютные. **Абсолютный путь** всегда начинается с корневого каталога и ведет вниз по иерархии до нужного файла. **Относительный путь** всегда начинается с **текущего каталога** и представляет путь к файлу относительно этого каталога. Например, давайте обратимся к файлу, который называется `poem` и находится в нашем домашнем каталоге. Если домашний каталог называется `packt` и вложен в каталог `home`, который в свою очередь вложен в корневой каталог (`/`), то абсолютный путь к файлу `poem` будет выглядеть так:

```
/home/packt/poem
```

¹ Этот популярный слоган стоит воспринимать с определенной долей условности, потому что в Linux все-таки существуют объекты, которые не являются файлами. Например, пользователи и группы (см. главу 4) — важнейшие сущности любой системы семейства Unix и Linux, но ни к какому отдельному пользователю или отдельной группе нельзя обратиться как к файлу. — *Примеч. ред.*

Если бы мы хотели вывести на экран содержимое этого файла, скажем, с помощью команды `cat` — можно было бы передать этой команде абсолютный путь:

```
cat /home/packt/poem
```

Относительный путь к тому же файлу зависит от того, какой каталог является текущим. В нашем случае, когда мы находимся в домашнем каталоге, `cat` можно запустить так:

```
cat poem
```

Полезно знать абсолютные пути к файлам, с которыми вы работаете. Немного попрактиковавшись, вы выучите пути к часто используемым файлам. Например, один из файлов, расположение которого стоит знать, — это `passwd`. Он находится в каталоге `/etc`, поэтому к нему можно обратиться по абсолютному пути `/etc/passwd`. Использовать относительный путь к файлу удобнее, если вы уже находитесь либо в том же каталоге, что и этот файл, либо где-то поблизости в файловой системе.

Чтобы работать с относительными путями, нужно знать два специальных обозначения, которые используются в путях к файлам. Одно из них — точка (`.`), которая обозначает текущий каталог, а другое — две последовательные точки (`..`), которые обозначают родительский каталог текущего каталога.

Оперируя относительными путями, всегда следите за тем, в каком каталоге вы находитесь. Чтобы узнать текущий каталог, используйте команду `pwd`.

Вот хороший пример того, как работать с относительными путями: допустим, вы находитесь внутри того или иного каталога и хотите обратиться к его родительскому каталогу. В следующем примере мы находимся в домашнем каталоге и хотим просмотреть список учетных записей системы, который хранится в файле `passwd`. Вот как нам поможет относительный путь:

```
packt@neptune:~$ pwd  
/home/packt
```

```
packt@neptune:~$ cat passwd  
cat: passwd: No such file or directory1
```

```
packt@neptune:~$ cat ../../etc/passwd
```

Здесь мы сначала запрашиваем текущий рабочий каталог командой `pwd`, и оболочка выводит путь к нашему домашнему каталогу `/home/packt`. Затем мы пытаемся вывести содержимое файла `passwd` с помощью команды `cat` прямо из домашнего каталога, но получаем сообщение об ошибке: в нашем домашнем каталоге нет файла или каталога с именем `passwd`. Проблема в том, что мы указали относительный путь, который всегда указывает на расположение относи-

¹ Нет такого файла или каталога. — *Примеч. пер.*

тельно текущего каталога. Чтобы избежать ошибки, мы используем последовательность из двух точек, которая ссылается на родительский каталог. В данном случае команда выглядит так: `cat ../../etc/passwd`.

Всегда используйте клавишу **Tab** для автозавершения путей, чтобы проверить, правильный ли путь вы вводите. В предыдущем примере мы набрали `../../etc` и нажали **Tab**, после чего автозавершение подставило косую черту. Затем мы набрали первые две буквы имени файла и снова нажали **Tab**, в результате чего получили список файлов в каталоге `/etc`, имена которых начинаются с `pa`. Увидев среди них искомый `passwd`, мы убедились, что путь указан верно, поэтому набрали еще два символа `s` и снова нажали **Tab**. Теперь команда готова, и можно нажать **Enter/Return**, чтобы ее запустить.

Путь в последней команде вычисляется относительно нашего домашнего каталога и читается так: вывести содержимое файла с именем `passwd`, который находится в каталоге `/etc`, который находится в родительском каталоге (первые две точки) родительского каталога (вторые две точки) текущего каталога (`home`). Таким образом, путь `../../etc/passwd`, указанный относительно нашего домашнего каталога, преобразуется в абсолютный путь `/etc/passwd`.

Далее мы поговорим о базовых операциях с файлами в Linux.

Базовые операции с файлами

Вам, как системному администратору, ежедневно приходится работать с файлами: создавать их, копировать, перемещать, удалять, создавать ссылки, формировать списки файлов и т. д. В этой главе мы уже упоминали некоторые команды для этих операций, но теперь пора подробнее остановиться на их использовании, ключах и атрибутах. Некоторые более сложные команды мы рассмотрим в следующих разделах.

Как создавать файлы

Бывают ситуации, когда нужно **создать** новый файл. Один из способов создать новый пустой файл — команда `touch`. Если ее вызвать, то появится новый файл: его владельцем будете вы, а его размер будет равен нулю, потому что он пустой.

В следующем примере мы создаем новый файл с именем `new-report` в каталоге `~/packt/`:

```
packt@neptune:~$ pwd
/home/packt
```

```
packt@neptune:~$ touch new-report
```

```
packt@neptune:~$ ls -l new-report
-rw-rw-r-- 1 packt packt 0 Mar  2 14:24 new-report
```

```
packt@neptune:~$ touch new-report
```

```
packt@neptune:~$ ls -l new-report
-rw-rw-r-- 1 packt packt 0 Mar  2 14:25 new-report
```

С помощью `touch` можно также изменить время последней модификации файла, не изменяя сам файл. Обратите внимание на разницу между начальным временем, когда мы впервые создали файл `new-report`, и новым временем после того, как мы применили к нему команду `touch`. А с помощью ключа `-a` команды `touch` можно изменить время последнего доступа к файлу. По умолчанию команда `ls` в расширенном формате выводит только время модификации (или создания) файла. Чтобы вывести время доступа, можно использовать ключ `-time` с параметром `atime`. Рассмотрим пример:

```
packt@neptune:~$ ls -l new-report
-rw-rw-r-- 1 packt packt 67 Mar  2 14:30 new-report
```

```
packt@neptune:~$ touch -a new-report
```

```
packt@neptune:~$ ls -l --time=atime new-report
-rw-rw-r-- 1 packt packt 67 Mar  2 14:31 new-report
```

Отметки времени модификации, создания и доступа к файлам очень полезны, особенно когда используются такие команды, как `find`, потому что эти данные позволяют детализировать поиск. Мы еще вернемся к этой команде на примерах в следующих разделах.

Чтобы создавать файлы, можно также перенаправлять вывод команды `echo`. По умолчанию она выводит строку, заданную в качестве аргумента, в стандартный вывод (то есть на экран). Но вывод команды `echo` можно перенаправить непосредственно в файл:

```
packt@neptune:~$ echo это демонстрация
это демонстрация
```

```
packt@neptune:~$ echo это тоже демонстрация > art-file
```

```
packt@neptune:~$ cat art-file
это тоже демонстрация
```

```
packt@neptune:~$ echo это еще одна демонстрация >> art-file
```

```
packt@neptune:~$ cat art-file
это тоже демонстрация
это еще одна демонстрация
```

В предыдущем примере мы перенаправили вывод из команды `echo` в файл `art-file`. До этого он не существовал, поэтому команда создала его автоматически. Первая команда `echo` заменила все содержимое файла (пустого на тот момент) на строку `это тоже демонстрация` с помощью оператора `>`, а вторая команда `echo`

добавила новую строку это еще одна демонстрация в конец файла с помощью оператора `>>`.

Как выводить список файлов

На протяжении этой книги мы уже приводили несколько примеров с командой `ls`, так что вы с ней немного знакомы. Также мы рассмотрели ключ `-l`, так что не будем изучать его снова, а перечислим новые ключи этой важной и полезной команды:

<code>-lh</code>	Ключ <code>-l</code> выводит список файлов в расширенном формате, а ключ <code>-h</code> показывает размер каждого файла в удобочитаемом виде — не в байтах, а в килобайтах или мегабайтах
<code>-a</code>	Перечислить все файлы, включая скрытые. Если использовать ключ <code>-a</code> вместе с <code>-l</code> , вы получите список всех файлов с подробными сведениями о них
<code>-tr</code>	Ключ <code>-t</code> сортирует файлы по времени их модификации (сначала новые), а ключ <code>-r</code> инвертирует порядок сортировки
<code>-S</code>	Сортировать файлы по размеру (сначала большие)
<code>-R</code>	Рекурсивно вывести содержимое указанного каталога и его подкаталогов

Команда `ls` часто используется с ключами `-la`, чтобы вывести список файлов в **расширенном формате**. Давайте подробнее поговорим об этом формате, хотя мы еще рассмотрим его во всех деталях в главе 4 «Как управлять пользователями и группами».

Мы уже выводили в расширенном формате содержимое нашего домашнего каталога на с. 66, когда сравнивали вывод команды `ls -la` с выводом `tree`. Вот еще один пример вывода:

```
total 48
drwxr-x--- 5 packt packt 4096 Mar  2 14:44 .
drwxr-xr-x 3 root  root  4096 Feb 27 08:58 ..
-rw-rw-r-- 1 packt packt   55 Mar  2 14:46 art-file
-rw----- 1 packt packt 1039 Mar  1 18:56 .bash_history
```

Здесь первая строка (**total 48**) показывает количество блоков в указанном каталоге. После этого каждая строка представляет один файл или подкаталог и содержит следующую подробную информацию:

- Первый символ — это тип файла: `d` — каталог; `-` — обычный файл; `l` — ссылка; `c` — символьное устройство; `b` — блочное устройство.
- Следующие 9 символов (например, `rwxr-x---`) — это права доступа к файлам (мы подробно рассмотрим их в главе 4 «Как управлять пользователями и группами»).
- Количество жестких ссылок на этот файл (см. «Как работать со ссылками» далее в этой главе).

- PID и GID владельца (например, `packt` — см. подробности в главе 4).
- Размер файла (например, `4096` — вид числа зависит от того, представлено ли оно в человекочитаемом формате).
- Время последней модификации файла.
- Имя файла или каталога.

Первые две строки после `total` — это ссылки на сам запрошенный каталог (точка `.`) и на его родительский каталог (две точки `..`).

В следующем разделе вы узнаете, как копировать и перемещать файлы.

Как копировать и перемещать файлы

Копировать файлы в Linux можно с помощью команды `cp`, а перемещать их по файловой системе — с помощью команды `mv`, которая также позволяет переименовывать файлы.

Чтобы скопировать файл, запустите `cp` в самой простой форме:

`cp что_копировать куда_копировать`

Здесь *что_копировать* — это путь к файлу, который нужно скопировать, а *куда_копировать* — путь к каталогу, в который копируется файл (и имя копии файла, если нужно). С помощью `cp` можно также скопировать несколько файлов в уже существующий каталог. Если каталога назначения не существует, оболочка сообщит вам, что целевой путь не является каталогом.

Давайте рассмотрим некоторые варианты этих команд:

- `cp -a`: ключ `-a` копирует всю иерархию каталогов в рекурсивном режиме, сохраняя все атрибуты и ссылки. В следующем примере мы копируем весь каталог `dir1`, который ранее создали в домашнем каталоге, во вновь созданный каталог `backup_dir1` с помощью `-a`:

```
packt@neptune:~$ ls
art-file  dir1  new-report  users

packt@neptune:~$ mkdir backup_dir1

packt@neptune:~$ ls
art-file  backup_dir1  dir1  new-report  users

packt@neptune:~$ cp -a dir1/ backup_dir1/

packt@neptune:~$ ls backup_dir1/
dir1
```

- `cp -r`: этот ключ действует так же, как `-a`, но сохраняет только символические ссылки, а не атрибуты.
- `cp -p`: ключ `-p` сохраняет права доступа и временные метки файлов. Если запускать `cp` в простейшей форме, без `-p`, то копии файлов будут принад-

лежать вашему пользователю, а их временные метки будут соответствовать тому моменту, когда вы выполняли операцию копирования.

- `cp -R`: ключ `-R` позволяет копировать каталог рекурсивно. В следующем примере мы используем команду `ls`, чтобы вывести содержимое каталога `~/packt/`, а затем команду `cp -R`, чтобы скопировать содержимое каталога `/files` в каталог `/new-files`. Этого каталога прежде не было, и команда `cp -R` создала его:

```
packt@neptune:~$ ls
art-file  backup_dir1  dir1  new-report  users

packt@neptune:~$ mkdir files
packt@neptune:~$ cd files/
packt@neptune:~/files$ touch files{1..6}
packt@neptune:~/files$ cd ..

packt@neptune:~$ ls
art-file  backup_dir1  dir1  files  new-report  users

packt@neptune:~$ cp -R files/ new-files
packt@neptune:~$ ls
art-file  backup_dir1  dir1  files  new-files  new-report  users

packt@neptune:~$ cd new-files/
packt@neptune:~/new-files$ ls
files1  files2  files3  files4  files5  files6
```

Перемещать файлы позволяет команда `mv`, с помощью которой можно либо перенести файлы и каталоги из одного места в другое, либо переименовать их. Вот как можно переименовать файл `files1` в `old-files1`: `mv files1 old-files1`.

Существует множество других возможностей, о которых вы узнаете, если просто обратитесь к страницам руководства. Не стесняйтесь изучать их и использовать в своих повседневных задачах.

Как работать со ссылками

Ссылки — очень привлекательная функция Linux. С их помощью можно защищать исходные файлы, а можно просто хранить несколько копий файла, выделяя место на диске только под одну копию. Рассматривайте ссылки как инструмент, который позволяет создавать альтернативные имена для одного и того же файла.

С помощью команды `ln` можно создавать два типа ссылок:

- символические;
- жесткие.

Эти типы ссылок соответствуют двум разным типам файлов, которые указывают на исходный файл. **Символическая ссылка** — это физический файл, который

указывает на исходный; файлы связаны друг с другом и имеют одинаковое содержимое. Кроме того, символическая ссылка может связывать разные файловые системы и физические носители, то есть ссылаться на исходные файлы, которые находятся на других дисках или на других разделах с другими типами файловых систем. Команда используется так:

```
ln -s исходный_файл ссылка
```

Вот пример, в котором мы выводим содержимое каталога `~/packt`, затем создаем символическую ссылку на файл `new-report` с помощью команды `ln -s`, а затем снова выводим содержимое:

```
packt@neptune:~$ ls
art-file backup_dir1 dir1 files new-files new-report users
```

```
packt@neptune:~$ ln -s new-report new-report-link
```

```
packt@neptune:~$ ls -l
total 28
-rw-rw-r-- 1 packt packt  55 Mar  2 14:46 art-file
drwxrwxr-x 3 packt packt 4096 Mar  2 15:20 backup_dir1
drwxrwxr-x 2 packt packt 4096 Mar  2 15:19 dir1
drwxrwxr-x 2 packt packt 4096 Mar  2 16:04 files
drwxrwxr-x 2 packt packt 4096 Mar  2 16:04 new-files
-rw-rw-r-- 1 packt packt  67 Mar  2 14:30 new-report
lrwxrwxrwx 1 packt packt  10 Mar  2 16:10 new-report-link -> new-report
-rw-rw-r-- 1 packt packt  67 Feb 27 14:33 users
```

Как видите, новая ссылка называется `new-report-link` и визуально представлена стрелкой `->`, которая указывает на исходный файл. Вы наверняка также заметили, что у ссылки и оригинального файла разные размеры и разные права доступа: это позволяет убедиться, что это два *разных* физических файла. Чтобы еще раз подтвердить это, можно запустить команду `ls -li`, которая покажет идентификатор **индексного дескриптора**¹ перед каждым файлом. В следующем примере видно, что у файлов `new-report` и `new-report-link` разные дескрипторы:

```
packt@neptune:~$ ls -li
total 28
137603 -rw-rw-r-- 1 packt packt  55 Mar  2 14:46 art-file
137621 drwxrwxr-x 3 packt packt 4096 Mar  2 15:20 backup_dir1
137604 drwxrwxr-x 2 packt packt 4096 Mar  2 15:19 dir1
137631 drwxrwxr-x 2 packt packt 4096 Mar  2 16:04 files
137638 drwxrwxr-x 2 packt packt 4096 Mar  2 16:04 new-files
137599 -rw-rw-r-- 1 packt packt  67 Mar  2 14:30 new-report
137849 lrwxrwxrwx 1 packt packt  10 Mar  2 16:10 new-report-link -> new-report
131094 -rw-rw-r-- 1 packt packt  67 Feb 27 14:33 users
```

¹ Индексный дескриптор (inode, на профессиональном жаргоне также айнод или инод) — структура данных, которая описывает объекты файловой системы (например, файлы и каталоги). У каждого индексного дескриптора есть уникальный числовой идентификатор. — *Примеч. ред.*

Если вы хотите узнать, на что указывает ссылка, не используя `ls -l`, воспользуйтесь командой `readlink`, которая доступна как в Ubuntu, так и в CentOS и выводит просто имя файла, на который указывает ссылка. Команда работает только с символическими ссылками:

```
packt@neptune:~$ readlink new-report-link
new-report
```

Этот вывод показывает, что файл `new-report-link` — это символическая ссылка на файл `new-report`.

В отличие от этого, **жесткая ссылка** — это виртуальный файл, который указывает на исходный файл. Физически оба файла одинаковы. Чтобы создать жесткую ссылку, используйте команду `ln` без каких-либо ключей:

ln исходный_файл ссылка

В следующем примере мы создаем жесткую ссылку на файл `new-report`, которая называется `new-report-ln`:

```
packt@neptune:~/links$ echo это исходный файл > new-report
packt@neptune:~/links$ cat new-report
это исходный файл

packt@neptune:~/links$ ln new-report new-report-ln
packt@neptune:~/links$ ls -l
total 8
-rw-rw-r-- 2 packt packt 17 Mar  2 16:23 new-report
-rw-rw-r-- 2 packt packt 17 Mar  2 16:23 new-report-ln

packt@neptune:~/links$ echo это жесткая ссылка >> new-report-ln

packt@neptune:~/links$ cat new-report-ln
это исходный файл
это жесткая ссылка

packt@neptune:~/links$ cat new-report
это исходный файл
это жесткая ссылка

packt@neptune:~/links$ ls -li
total 8
138839 -rw-rw-r-- 2 packt packt 34 Mar  2 16:24 new-report
138839 -rw-rw-r-- 2 packt packt 34 Mar  2 16:24 new-report-ln
```

Как видите, у обоих файлов одинаковый размер и один и тот же индексный дескриптор. А после того, как мы изменили исходный файл с помощью `echo` и перенаправления вывода, изменения отразились на обоих файлах. Эти файлы представлены не так, как символические ссылки: в выводе `ls` они отображаются как два разных файла, и ничто не указывает на то, какой файл ссылается на какой. По сути жесткая ссылка связана с индексным дескриптором исходного файла. Можно считать, что для файла создается дополнительное имя; это похоже на переименование, но все-таки не то же самое.

Как удалять файлы

Удалять файлы в Linux позволяет команда `rm`, которая в своей простейшей форме используется без ключей. Чтобы полнее контролировать удаление элементов, можно добавить ключи `-i`, `-f` и `-r`:

- `rm -i` включает интерактивный режим, запрашивая согласие перед тем, как удалить файл:

```
packt@neptune:~$ ls
art-file  dir1  links  new-report  users
backup_dir1  files  new-files  new-report-link
```

```
packt@neptune:~$ rm -i art-file
rm: remove regular file 'art-file'?1 y
```

```
packt@neptune:~$ ls
backup_dir1  dir1  files  links  new-files  new-report  new-report-link
users
```

В предыдущем примере мы удалили файл `art-file` с ключом `-i`. Когда команды такого рода выводят запрос, у вас есть два варианта: можно одобрить действие, нажав `y` («да»), или отменить его, нажав `n` («нет»).

- `rm -f` удаляет файл принудительно, без взаимодействия с пользователем:

```
packt@neptune:~$ ls
backup_dir1  dir1  files  links  new-files  new-report  new-report-link
users
```

```
packt@neptune:~$ rm -f new-report-link
```

```
packt@neptune:~$ ls
backup_dir1  dir1  files  links  new-files  new-report  users
```

Мы удалили созданный ранее файл `new-report-link` с помощью команды `rm -f`. Она не спрашивала нашего согласия и удалила файл принудительно.

- `rm -r` удаляет файлы в рекурсивном режиме и позволяет удалить несколько файлов и каталогов. Например, если попытаться удалить каталог `new-files` командой `rm` без ключей, она выведет сообщение об ошибке, а именно о том, что невозможно удалить каталог. Но если использовать ключ `-r`, каталог будет немедленно удален:

```
packt@neptune:~$ ls
backup_dir1  dir1  files  links  new-files  new-report  users
```

```
packt@neptune:~$ rm new-files/
rm: cannot remove 'new-files/': Is a directory2
```

¹ Удалить обычный файл `art-file`? — *Примеч. пер.*

² Не удается удалить `'new-files'`: это каталог. — *Примеч. пер.*

```
packt@neptune:~$ rm -r new-files/  
packt@neptune:~$ ls  
backup_dir1  dir1  files  links  new-report  users
```

Будьте *особенно осторожны*, когда используете команду `rm`. Самый разрушительный режим — команда `rm -rf`, которая рекурсивно удаляет файлы, каталоги и все остальное без предупреждения. Имейте в виду, что это необратимое действие, которое может нанести огромный ущерб¹.

Чаще всего удаление файлов — это дорога в один конец, и откатить это действие невозможно. Поэтому очень важно следить за тем, что вы удаляете, а если перед удалением создавать резервные копии, это может избавить вас от лишней головной боли.

Как создавать каталоги

В Linux можно создать новый каталог с помощью команды `mkdir`. В следующем примере мы создаем новый каталог `development`:

```
mkdir development
```

Если нужно создать сразу цепочку вложенных каталогов, используйте ключ `-p`, как показано здесь:

```
packt@neptune:~$ mkdir -p reports/month/day
```

```
packt@neptune:~$ tree reports/  
reports/  
├── month  
│   └── day
```

```
2 directories, 0 files
```

Каталоги в Linux — это тоже файлы, только с особыми атрибутами. Они необходимы, чтобы структурировать файловую систему. Чтобы подробнее узнать о команде `mkdir`, обратитесь к справочным страницам.

Как удалять каталоги

Удалять каталоги в Linux можно с помощью команды `rmdir`, которая по умолчанию удаляет только пустые каталоги. Давайте посмотрим, что произойдет, если попробовать удалить непустой каталог:

¹ Много лет назад команда `rm` была особенно вредоносна в форме `rm -rf /`, потому что при этом удалялся корневой каталог (то есть фактически все содержимое файловой системы) и система становилась неработоспособной. В современные системы семейства Linux встроена защита от удаления корневого каталога. Если вы *действительно* хотите его удалить, запустите команду `rm -rf --no-preserve-root /`. — *Примеч. ред.*

```
packt@neptune:~$ rmdir reports/  
rmdir: failed to remove 'reports/': Directory not empty1
```

Это мера предосторожности со стороны оболочки, потому что удаление непустого каталога может привести к плачевным последствиям, как мы уже видели на примере команды `rm`. У команды `rmdir` нет опции `-r`, как у `rm`, поэтому, чтобы удалить каталог с помощью `rmdir`, придется сначала вручную удалить файлы внутри него. Однако команда `rm -r`, показанная ранее, по-прежнему полезна и более универсальна при удалении каталогов.

Теперь, когда вы знаете, как работать с каталогами в Linux, перейдем к командам для просмотра файлов.

Как просматривать файлы

Поскольку в Linux все является файлом, всякому системному администратору необходимо уметь просматривать содержимое файла и работать с ним. В этом разделе мы изучим команды для просмотра файлов, потому что почти все файлы содержат текст, который в какой-то момент можно прочитать.

Команда `cat`

Мы использовали эту команду в нескольких примерах ранее в этой главе. Название команды — сокращение от *concatenate*²; она выводит содержимое файла на экран. Вот еще один пример: допустим, у нас есть два файла, один из которых называется `new-report`, а другой — `users`. Вот как просмотреть их содержимое с помощью `cat`:

```
packt@neptune:~$ cat new-report  
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor  
incididunt ut labore et dolore magna aliqua.
```

```
packt@neptune:~$ cat new-report users  
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor  
incididunt ut labore et dolore magna aliqua.  
packt    pts/0    2023-02-27 13:49 (192.168.124.1)  
/home/packt
```

В этом примере мы сначала запустили `cat`, чтобы показать содержимое только одного файла `new-report`. А вторая команда вывела содержимое сразу двух файлов — `new-report` и `users`. В данном случае оба файла находятся в одном каталоге, который также является текущим рабочим каталогом. Если нужно вывести содержимое файла, который находится в другом каталоге, укажите абсолютный путь к нему.

¹ Не удается удалить 'reports/': каталог не пуст. — *Примеч. пер.*

² Сцеплять, связывать. — *Примеч. пер.*

У команды `cat` есть несколько ключей, которые мы не будем здесь рассматривать, потому что чаще всего она запускается без них. Более подробную информацию можно найти на справочных страницах.

Команда `less`

Бывают случаи, когда в файле так много текста, что он занимает не один экран, и его трудно просмотреть на терминале, если использовать только `cat`. Вот тут-то и пригодится команда `less`, которая показывает один экран за раз. Сколько именно строк выводится — это зависит от размера окна терминала. Возьмем, например, файл `/etc/passwd`, все строки которого могут не уместиться на одном экране. Его можно просмотреть с помощью такой команды:

```
$ less /etc/passwd
```

Когда вы нажмете `Enter`, на экране появится содержимое файла. Перемещаться по нему можно с помощью таких клавиш:

Клавиша пробел	На один экран вперед
Enter	На одну строку вперед
b	На один экран назад
/	Переключиться в режим поиска; при этом выполняется поиск вперед по файлу
?	Режим поиска; выполняется поиск назад по файлу
v	Редактировать файл с помощью редактора по умолчанию
g	В начало файла
Shift + g	В конец файла
q	Выход из просмотра

У команды `less` немало полезных ключей. Обратитесь к справочным страницам, чтобы узнать о ней подробнее.

Команда `head`

Эта команда удобна, когда нужно вывести на экран только начало текстового файла. По умолчанию она выводит первые 10 строк. Давайте поэкспериментируем с `head` на примере того же файла `/etc/passwd`. Посмотрите, что произойдет, если запустить следующую команду. Она напечатает первые 10 строк, а затем завершит работу и вернет вас в приглашение оболочки:

```
head /etc/passwd
```

Полезное свойство команды `head` состоит в том, что она умеет выводить больше или меньше десяти строк файла. Для этого можно использовать ключ `-n` или

просто - (дефис), после которого указать нужное количество строк. В следующем примере мы сначала вызываем команду `head` для файла `/etc/passwd` без каких-либо ключей, а затем указываем количество строк:

```
packt@neptune:~$ head /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin

packt@neptune:~$ head -n 3 /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin

packt@neptune:~$ head -3 /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
```

У этой команды есть много других возможностей, которые могут пригодиться в вашей работе в качестве системного администратора, но здесь мы не будем их рассматривать. Надеемся, что вы изучите подробности самостоятельно.

Команда `tail`

Команда `tail` похожа на `head`, только по умолчанию она печатает последние 10 строк файла. Чтобы увидеть определенное количество строк от конца файла, можно использовать тот же аргумент `-n`, что и для команды `head`, однако чаще всего `tail` используют для того, чтобы в режиме реального времени просматривать постоянно изменяющиеся файлы журнала. Эта команда может вывести последние строки файла и тогда, когда в него пишут другие приложения. Рассмотрим пример:

```
tail -f /var/log/syslog
```

Ключ `-f` заставляет команду следить за файлом `/var/log/syslog` по мере того, как он дополняется новыми записями; при этом новые строки будут отображаться на экране в режиме реального времени. Команда `tail` с ключом `-f` прекратит работу во время ротации журнала, но если использовать ключ `-F`, то вывод продолжит отображаться даже во время ротации. Чтобы выйти из режима просмотра и вернуться в приглашение командной оболочки, нажмите `Ctrl + C`. Вот пример того, что выводит предыдущая команда:

```
packt@neptune:~$ tail -f /var/log/syslog
Mar  2 18:54:26 neptune systemd[1]: Starting Time & Date Service...
Mar  2 18:54:27 neptune dbus-daemon[643]: [system] Successfully activated servic
```

```
e 'org.freedesktop.timedate1'
Mar  2 18:54:27 neptune systemd[1]: Started Time & Date Service.
Mar  2 18:54:27 neptune snapd[653]: storehelpers.go:769: cannot refresh: snap has no updates available: "core20", "lxd", "snapd"
Mar  2 18:54:27 neptune snapd[653]: autorefresh.go:551: auto-refresh: all snaps are up-to-date
Mar  2 18:54:57 neptune systemd[1]: systemd-timedated.service: Deactivated successfully.
Mar  2 19:04:26 neptune systemd[1]: Starting Cleanup of Temporary Directories...
Mar  2 19:04:26 neptune systemd[1]: systemd-tmpfiles-clean.service: Deactivated successfully.
Mar  2 19:04:26 neptune systemd[1]: Finished Cleanup of Temporary Directories.
Mar  2 19:17:01 neptune CRON[1081]: (root) CMD ( cd / && run-parts -- report /etc/cron.hourly)
Mar  2 19:27:39 neptune systemd[1]: Started Session 4 of User packt.
Mar  2 19:27:47 neptune systemd[1]: Starting Update APT News...
Mar  2 19:27:47 neptune systemd[1]: Starting Update the local ESM caches...
Mar  2 19:27:48 neptune systemd[1]: esm-cache.service: Deactivated successfully.
Mar  2 19:27:48 neptune systemd[1]: Finished Update the local ESM caches.
Mar  2 19:27:48 neptune systemd[1]: apt-news.service: Deactivated successfully.
Mar  2 19:27:48 neptune systemd[1]: Finished Update APT News.
```

Далее мы узнаем, как просмотреть свойства файла в Linux.

Как работать со свойствами файлов

Бывают случаи, когда просто просмотреть содержимое файла недостаточно, и нужны дополнительные сведения о нем. Для этого есть несколько удобных команд, которые мы обсудим в следующих разделах.

Команда *stat*

Команда *stat* дает больше информации, чем *ls*. В следующем примере можно сравнить вывод команд *ls* и *stat* для одного и того же файла:

```
packt@neptune:~$ ls
backup_dir1  development  dir1  files  links  new-report  reports  users

packt@neptune:~$ ls -l new-report
-rw-rw-r-- 1 packt packt 124 Mar  2 19:05 new-report

packt@neptune:~$ stat new-report
  File: new-report
  Size: 124          Blocks: 8          IO Block: 4096    regular file
Device: fd00h/64768d Inode: 137599   Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 1000/   packt)   Gid: ( 1000/   packt)
Access: 2023-03-02 19:06:00.455577063 +0000
Modify: 2023-03-02 19:05:55.243562331 +0000
Change: 2023-03-02 19:05:55.243562331 +0000
 Birth: 2023-03-02 14:24:14.834170882 +0000
```

Команда *stat* предоставляет дополнительную информацию о файле: имя (File), размер (Size), количество блоков (Blocks), тип файла (в данном слу-

чае *regular file* — обычный файл), индексный дескриптор (*Inode*), количество ссылок (*Links*), права доступа (первый *Access*), идентификаторы пользователя и группы владельца файла (*UID* и *GID*), а также время последнего доступа (второй *Access*, или *atime*), время последней модификации файла (*Modify*, или *mtime*) и время последнего изменения атрибутов файла (*Change*, или *ctime*). Чтобы подробнее узнать о команде *stat*, обратитесь к справочным страницам.

Команда *file*

Эта команда просто выводит тип файла. Вот пример вывода *stat* для текстового и исполняемого файла:

```
packt@neptune:~$ file new-report
new-report: ASCII txt
```

```
packt@neptune:~$ file /usr/bin/wh
whatis which whiptail whoami
whereis which.debianutils who
```

```
packt@neptune:~$ file /usr/bin/who
/usr/bin/who: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamical
ly linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=98fb99c65957c7
4a5216e69b3dd5032df976a2dc, for GNU/Linux 3.2.0, stripped
```

```
packt@neptune:~$ file /usr/bin/which
/usr/bin/which: symbolic link to /etc/alternatives/which
```

```
packt@neptune:~$ file /usr/bin/whoami
/usr/bin/whoami: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynami
cally linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=a18184f5d26
fce82fe5ccf842d3cf7b9c729030f, for GNU/Linux 3.2.0, stripped
```

В отличие от некоторых других операционных систем Linux не полагается на расширения и MIME-типы файлов. Команда *file* определяет тип файла скорее по его содержимому, чем по каким-то еще признакам.

Владельцы файлов и права доступа к файлам

В Linux **безопасность файлов** регулируется тем, кому принадлежит файл, а **принадлежность файла** определяется его владельцем и группой владельца. По отношению к каждому файлу все пользователи делятся на три категории: *пользователь (user)*, *группа (group)* и *все остальные (other)*. Пользователь — это чаще всего владелец файла, то есть по умолчанию тот, кто создал файл. Владельца можно сменить с помощью команды *chown*. Задавая группу владельца, вы определяете права доступа для всех ее членов — это настраивается с помощью команды *chgrp*. Под всеми остальными подразумеваются все прочие пользователи в этой системе — то есть те, кто не создавал файл, не является его владельцем и не принадлежит к группе владельца. Эту категорию еще называют «весь мир».

Система должна не только устанавливать принадлежность файлов, но и регулировать доступ пользователей к ним, а для этого служат **права доступа**, или **разрешения**. Рассмотрим свойства файлов в перечне, который выводит команда `ls -l`:

```
packt@neptune:~$ ls -l
total 32
drwxrwxr-x 3 packt packt 4096 Mar  2 15:20 backup_dir1
drwxrwxr-x 2 packt packt 4096 Mar  2 18:51 development
drwxrwxr-x 2 packt packt 4096 Mar  2 15:19 dir1
drwxrwxr-x 2 packt packt 4096 Mar  2 16:04 files
drwxrwxr-x 2 packt packt 4096 Mar  2 16:23 links
-rw-rw-r-- 1 packt packt 124 Mar  2 19:05 new-report
drwxrwxr-x 3 packt packt 4096 Mar  2 18:52 reports
-rw-rw-r-- 1 packt packt  67 Feb 27 14:33 users
```

В предыдущем примере вы видите всего два разных набора прав доступа для файлов в домашнем каталоге. В каждой строке вывода предусмотрены 12 позиций для символов, которые обозначают специальные атрибуты и права доступа, хотя до сих пор в примерах эти символы занимали только 10 позиций (например, `drwxrwxr-x`). Здесь первый символ обозначает тип файла, а остальные девять — права доступа, которым соответствуют такие знаки:

	Сокращение от	Право доступа
-	—	Нет доступа
r	read	Чтение
w	write	Запись
x	execute	Выполнение

Эти девять символов делятся на три блока по три символа: первый блок обозначает права владельца, второй — права группы-владельца, а последний — права остальных пользователей («всего мира»).

Типы файлов тоже обозначаются символами:

-	Обычный файл
d	Каталог
l	Символическая ссылка
p	Именованный канал — специальный файл, который облегчает обмен данными между программами
s	Сокет — файл, похожий на канал, но с двунаправленным и сетевым обменом данными
b	Блочное устройство — файл, соответствующий аппаратному устройству
c	Символьное устройство, подобно блочному

Каждую трехсимвольную группу прав доступа можно выразить **восьмеричным числом** (потому что восьмеричное число состоит из трех байт). Таким образом, права представляются как степени двойки:

Чтение	r	$2^2 = 4$
Запись	w	$2^1 = 2$
Выполнение	x	$2^0 = 1$

В результате права доступа к файлам можно представить как сумму чисел по следующей схеме:

	r		w		x		r		w		x		r		w		x			
владелец			группа			остальные														
	чтен.		зап.		вып.		чтен.		зап.		вып.		чтен.		зап.		вып.			
	4		2		1		4		2		1		4		2		1			

На предыдущей диаграмме права представлены в виде строки из девяти символов, как в выводе `ls -la`. Символы разделены на три равных блока: первый — для владельца, второй — для группы-владельца, а третий — для остальных пользователей, как показано на второй строке. В двух остальных строках изображены типы разрешений (чтение, запись и выполнение) и восьмеричные числа, о которых пойдет речь в следующем абзаце.

Эта схема связывает восьмеричные представления прав доступа с символическими представлениями. Например, правам `rwX r-x r-x` соответствует восьмеричное число `755`. Почему это так? У владельца файла есть полные права доступа (первый блок `rwX`), что выражается как $4 + 2 + 1 = 7$. У группы-владельца есть права только на чтение и выполнение (второй блок `r-x`), что выражается как $4 + 1 = 5$. Наконец, остальным пользователям соответствуют такие же права (третий блок `r-x`), что тоже выражается как $4 + 1 = 5$. Таким образом, правам доступа к файлу соответствует восьмеричное число `755`.

В качестве упражнения попробуйте перевести в восьмеричную систему такие права:

- `rwX rwX rwX`
- `rwX r-x r--`
- `rwX r-x ---`
- `rwX --- ---`
- `rw- rw- rw-`
- `rw- rw- r--`
- `rw- rw- ---`
- `rw- r-- r--`

- `rw- r-- ---`
- `rw- --- ---`
- `r-- --- ---`

Есть и другие важные команды — такие, как `umask`, `chown`, `chmod` и `chgrp`, которые позволяют изменить или задать режим создания по умолчанию, владельца, права доступа и группу соответственно. Здесь мы упоминаем их вкратце, потому что они относятся к настройке свойств файла, а подробнее мы рассмотрим эти команды в главе 4 «Как управлять пользователями и группами».

Сжатие и архивация файлов

В Linux стандартный инструмент для архивации называется `tar` (от *tape archive*¹). Изначально его использовали в Unix для того, чтобы архивировать файлы на внешних ленточных накопителях. В современных системах семейства Linux он также позволяет сохранять файлы в сжатом формате. Кроме формата `tar`, к популярным форматам сжатия данных относятся `gzip` и `bzip`, а также широко известный `zip` из Windows. Давайте рассмотрим команду `tar` подробнее.

Команда `tar` для сжатия и распаковки файлов

По умолчанию (без ключей) эта команда не сжимает данные. Чтобы применить сжатие и настроить другие параметры, нужно использовать ключи, самые полезные из которых мы перечислим здесь:

<code>-c</code>	Создать архив
<code>-r</code>	Добавить файлы в уже существующий архив
<code>-u</code>	Добавить только измененные файлы в существующий архив
<code>-A</code>	Добавить архив в конец другого архива
<code>-t</code>	Вывести список содержимого архива
<code>-x</code>	Извлечь содержимое архива
<code>-z</code>	Сжимать данные в формате <code>gzip</code>
<code>-j</code>	Сжимать данные в формате <code>bzip2</code>
<code>-v</code>	Вывести подробную информацию о ходе выполнения
<code>-p</code>	Восстановить исходные права доступа и владельцев для извлеченных файлов
<code>-f</code>	Задать имя выходного файла

Весьма вероятно, что в повседневной работе вам придется использовать эти аргументы в сочетании друг с другом.

¹ Архив на магнитной ленте. — *Примеч. пер.*

Например, чтобы поместить в архив каталог `files`, можно совместно использовать аргументы `-cvf`:

```
packt@neptune:~$ ls
backup_dir1  development  dir1  files  links  new-report  reports  users

packt@neptune:~$ tar -cvf files-archive.tar files/
files/
files/files4
files/files1
files/files5
files/files6
files/files3
files/files2

packt@neptune:~$ ls
backup_dir1  dir1  files-archive.tar  new-report  users
development  files  links  reports
```

Мы создали архив без сжатия. Чтобы сжать данные, нужно добавить аргументы `-z` или `-j`. В следующем примере мы используем ключ `-z`, чтобы применить алгоритм сжатия `gzip`. Сравните размер двух архивных файлов. Как правило, в таких случаях рекомендуется снабжать файлы расширениями:

```
packt@neptune:~$ tar -czvf files-archive-gzipped.tar.gz files
files/
files/files4
files/files1
files/files5
files/files6
files/files3
files/files2

packt@neptune:~$ ls -l files-archive-gzipped.tar.gz files-archive.tar
-rw-rw-r-- 1 packt packt 189 Mar  2 20:08 files-archive-gzipped.tar.gz
-rw-rw-r-- 1 packt packt 10240 Mar  2 20:04 files-archive.tar
```

Чтобы распаковать архив в формате `tar`, можно использовать ключ `-x`, который показан в предыдущей таблице. Например, распакуем файл `files-archive.tar`, который мы создали ранее, а также с помощью ключа `-C` укажем, в какой каталог помещать распакованные файлы. Этот каталог нужно создать заранее:

```
mkdir uncompressed-directory
tar -xvf files-archive.tar -C uncompressed-directory
```

Это позволит извлечь файлы из архива и добавить их в каталог `uncompressed-directory`. Чтобы распаковать архив, сжатый с помощью `gzip` (например, `files-archive-gzipped.tar.gz`), добавим ключ `-z` к тем, что фигурировали в предыдущей команде:

```
tar -xvzf files-archive-gzipped.tar.gz -C uncompressed-directory
```

Теперь вы знаете, как архивировать и распаковывать файлы в Linux. В этой системе есть и другие полезные инструменты для архивации, но `tar` по-прежнему используется чаще всего. Не поленитесь изучить альтернативные инструменты, а также прочие полезные ключи `tar`.

Поиск файлов

Искать файлы в Linux — важная задача для любого системного администратора. Поскольку в системах Linux чрезвычайно много файлов, поиск нужного файла может оказаться непростым делом. Однако в вашем распоряжении есть удобные инструменты, и, если вы умеете с ними обращаться, это станет одним из ваших главных преимуществ. В следующих разделах мы рассмотрим такие команды, как `locate`, `which`, `whereis` и `find`.

Команда `locate`

Команда `locate` не установлена по умолчанию в Ubuntu. Чтобы ее установить, используйте такую команду, которая проиндексирует все расположения файлов в вашей системе:

```
sudo apt install mlocate1
```

После этого, если запустить команду `locate`, она будет искать файл в своей базе данных, используя команду `updatedb` в качестве вспомогательного инструмента.

Прежде чем начать использовать команду `locate`, нужно запустить `updatedb`, чтобы обновить индекс. После этого для примера найдем все файлы, в именах которых есть `new-report`:

```
packt@neptune:~$ locate new-report
/home/packt/new-report
/home/packt/links/new-report
/home/packt/links/new-report-ln
```

Если бы мы искали файл по менее специфическому имени — например, `presentation`, — то вывод получился бы слишком длинным и бесполезным. Вот пример, где мы перенаправили вывод в файл и использовали команду `wc` (которая подсчитывает количество слов), чтобы вывести в стандартный вывод только количество строк, слов и байт в файле:

```
packt@neptune:~$ locate presentation > /home/packt/locate-search && wc /home/pack
kt/locate-search
  8   8 663 /home/packt/locate-search
```

¹ Пакет `mlocate` в последний раз обновлялся в 2012 году и по умолчанию недоступен во многих современных системах. Вместо него используется `plocate` (plocate.sesse.net), который работает существенно быстрее и требует меньше дискового пространства для индекса. — *Примеч. ред.*

```
packt@neptune:~$ cat locate-search
/usr/share/mime/application/vnd.ms-powerpoint.presentation.macroenabled.12.xml
/usr/share/mime/application/vnd.oasis.opendocument.presentation-flat-xml.xml
/usr/share/mime/application/vnd.oasis.opendocument.presentation-template.xml
/usr/share/mime/application/vnd.oasis.opendocument.presentation.xml
/usr/share/mime/application/vnd.openxmlformats-officedocument.presentationml.presentation.xml
/usr/share/mime/application/vnd.openxmlformats-officedocument.presentationml.slide.xml
/usr/share/mime/application/vnd.openxmlformats-officedocument.presentationml.slide.show.xml
/usr/share/mime/application/vnd.openxmlformats-officedocument.presentationml.template.xml
```

В результирующем файле из предыдущего вывода восемь строк. Это значит, что мы нашли восемь файлов, в имени которых содержится строка `presentation`. Команда `wc` насчитала в файле столько же слов, потому что внутри путей к файлам нет пробелов, а значит, каждая строка воспринимается как одно слово. Кроме того, мы видим, что результирующий файл занимает 663 байта. Попробуйте сами поэкспериментировать с другими строками. О дополнительных ключах команды `locate` можно узнать на справочных страницах Linux.

Команда `which`

Эта команда обнаруживает местоположение исполняемого файла (программы или команды) в пути поиска оболочки. Например, чтобы узнать, где находится команда `ls`, введите следующее:

```
packt@neptune:~$ which ls
/usr/bin/ls
```

Как видите, выводится путь к исполняемому файлу команды `ls`: `/usr/bin/ls`.

Теперь попробуем так же запросить путь для команды `cd`:

```
which cd
```

Вы увидите, что ничего не выводится. Это потому, что команда `cd` встроена в оболочку и у нее нет отдельного исполняемого файла.

Команда `whereis`

Эта команда находит *только* исполняемые файлы, файлы документации и файлы исходного кода. Если вы ищете что-то другое, `whereis` вам не поможет, поэтому используйте ее с осторожностью:

```
packt@neptune:~$ whereis ls
ls: /usr/bin/ls /usr/share/man/man1/ls.1.gz
```

```
packt@neptune:~$ whereis cd
cd:
```

И снова для команды `cd` вывод ничего не показывает, потому что это встроенная команда оболочки. Зато для команды `ls` выводится расположение самой команды и ее справочных страниц.

Команда `find`

Это одна из самых мощных команд в Linux. Она позволяет искать файлы в каталогах и подкаталогах по определенным критериям и поддерживает более 50 ключей. Ее главный недостаток — синтаксис, который несколько отличается от других команд Linux. В том, как работает `find`, лучше всего разобраться на примерах, которые мы вскоре рассмотрим. Чтобы подробнее ознакомиться с мощными возможностями этой команды, обратитесь к справочным страницам.

Вот внушительный список упражнений с командой `find`, которые, по нашему мнению, пригодятся вам на практике. Мы демонстрируем сами команды, но не цитируем то, что они выводят, потому что некоторые результаты могут быть весьма объемными.

- Найти рекурсивно¹ в корневом каталоге все файлы, которые называются `e100`, и вывести их в стандартный вывод:

```
sudo find / -name e100 -print
```

- Найти в корневом каталоге все обычные файлы (`-type f`), в имени которых содержится строка `file`, и вывести их в стандартный вывод:

```
sudo find / -name "*file*" -type f -print
```

- Найти только в каталогах `/opt`, `/usr` и `/var` все обычные файлы, в имени которых содержится строка `print`:

```
sudo find /opt /usr /var -name "*print*" -type f -print
```

- Найти в корневом каталоге все обычные файлы с расширением `.conf`:

```
sudo find / type f -name "*.conf"
```

- Найти в корневом каталоге все обычные файлы, имя которых начинается с `file` и у которых есть расширение:

```
sudo find / -type f -name "file*.*"
```

- Найти в корневом каталоге все обычные файлы с расширениями `.c`, `.sh` и `.py` и добавить их список в файл `findfile`:

```
sudo find / -type f \( -name "*.c" -o -name "*.sh" -o -name  
"*.py" \) > findfile
```

¹ Здесь и далее предполагается рекурсивный поиск по всему содержимому каталога. — *Примеч. ред.*

- Найти в корневом каталоге все обычные файлы с расширением .c, отсортировать их список и добавить в файл:

```
sudo find / -type f -name "*.c" -print | sort > findfile2
```

- Найти в корневом каталоге все обычные файлы с правами доступа 0664:

```
sudo find / -type f -perm 0664
```

- Найти в корневом каталоге все обычные файлы, которые доступны их владельцу для чтения:

```
sudo find / -type f -perm /u=r
```

- Найти в корневом каталоге все исполняемые файлы:

```
sudo find / -type f -perm /a=x
```

- Найти в корневом каталоге все обычные файлы, которые были изменены позавчера:

```
sudo find / -type f -mtime 2
```

- Найти в корневом каталоге все обычные файлы, к которым последний раз обращались позавчера:

```
sudo find / -type f -atime 2
```

- Найти в корневом каталоге все обычные файлы, которые последний раз изменялись от 2 до 5 дней назад:

```
sudo find / -type f -mtime +2 -mtime -5
```

- Найти в корневом каталоге все обычные файлы, которые изменялись за последние 10 минут:

```
sudo find / -type f -mmin -10
```

- Найти в корневом каталоге все обычные файлы, которые были созданы за последние 10 минут:

```
sudo find / -type f -cmin -10
```

- Найти в корневом каталоге все обычные файлы, к которым обращались в последние 10 минут:

```
sudo find / -type f -amin -10
```

- Найти в корневом каталоге все обычные файлы размером 5 Мбайт:

```
sudo find / -type f -size 5M
```

- Найти в корневом каталоге все обычные файлы размером от 5 до 10 Мбайт:

```
sudo find / -type f -size +5M -size -10M
```

- Найти все пустые файлы и каталоги:

```
sudo find / -type f -empty  
sudo find / -type d -empty
```

- Найти пять самых больших файлов в каталоге `/etc`. Обратите внимание, что эта команда может быть очень ресурсоемкой. Не стоит запускать ее для всего корневого каталога, потому что вы рискуете исчерпать системную память:

```
sudo find /etc -type f -exec ls -s {} \; | sort -n -r | head -5
```

- Найти пять самых маленьких файлов в каталоге `/etc`¹:

```
sudo find /etc -type f -exec ls -s {} \; | sort -n | head -5
```

Не стесняйтесь экспериментировать с разными параметрами поиска. Команда `find` очень универсальна и мощна, так что ее стоит применять с осторожностью.

Работа с текстом в Linux

Работа с текстом — это, пожалуй, одна из самых сильных сторон Linux. В этой системе есть множество инструментов для того, чтобы работать с текстом в командной строке. Среди самых важных и популярных инструментов — `grep`, `tee`, а также более мощные команды, такие как `sed` и `awk`. Впрочем, мы вернемся к ним в главе 8, когда пойдет речь о том, как создавать и использовать сценарии. В этом разделе мы лишь вкратце упомянем, как применять эти инструменты в командной строке.

Команда `grep`

Это одна из самых мощных и при этом самых полезных команд в Linux. Она позволяет искать строки внутри текстовых файлов и поддерживает широкий ассортимент ключей:

<code>-v</code>	Показать строки, которые не соответствуют критериям поиска
<code>-l</code>	Показать только имена файлов, которые соответствуют критериям
<code>-L</code>	Показать только имена файлов, которые не соответствуют критериям
<code>-c</code>	Показать только количество строк, которые соответствуют критериям
<code>-n</code>	Показать номера строк, в которых найдены совпадения
<code>-i</code>	Искать без учета регистра
<code>-r</code>	Рекурсивный поиск по дереву каталогов
<code>-R</code>	Рекурсивный поиск по дереву каталогов (аналогично <code>-r</code>), при этом поиск переходит по всем символическим ссылкам
<code>-E</code>	Использовать расширенные регулярные выражения
<code>-F</code>	Считать шаблоны строковыми литералами, а не регулярными выражениями

¹ Если в каталоге `/etc` больше пяти одинаково маленьких файлов (например, если там 20 файлов нулевого размера), то эта команда выведет имена первых пяти файлов в ASCII-алфавитном порядке. — *Примеч. ред.*

Вот несколько примеров того, как можно использовать команду **grep**:

- Узнать, когда использовалась команда **sudo**:

```
sudo grep sudo /var/log/auth.log
```
- Искать строку **packt** в текстовых файлах из каталога **/etc** (со всеми подкаталогами):

```
grep -R packt /etc
```
- Показать номера строк, в которых найдено совпадение:

```
grep -Rn packt /etc
```
- Если вы не хотите видеть имена файлов, в которых найдено совпадение, используйте ключ **-h**. Тогда **grep** покажет только подходящие строки:

```
grep -Rh packt /etc
```
- Наоборот, чтобы вывести только имена файлов, в которых найдено совпадение, используйте ключ **-l**:

```
grep -Rl packt /etc
```

В большинстве случаев **grep** используют в сочетании с неименованными каналами оболочки. Вот несколько примеров:

- Если вам нужен список одних только каталогов из текущего рабочего каталога, вывод команды **ls** можно перенаправить в **grep**. В следующем примере выводятся только строки, которые начинаются с буквы **d**, а значит, представляют каталоги:

```
ls -la | grep '^d'
```
- Чтобы вывести модель вашего процессора, можно воспользоваться такой командой:

```
cat /proc/cpuinfo | grep -i 'Model'
```

Команда **grep** станет одним из самых востребованных инструментов в вашей работе системного администратора Linux, поэтому не бойтесь копаться в ее ключах и расширенных возможностях.

Команда **tee**

Эта команда очень похожа на **cat**. По сути, она делает то же самое: копирует стандартный ввод в стандартный вывод без изменений, однако при этом также копирует его в один или несколько файлов.

В следующем примере мы подсчитываем количество строк в файле **/etc/passwd/** с помощью команды **wc**, а затем перенаправляем вывод в команду **tee** с ключом **-a** (добавить в конец файла, если он уже существует), которая записывает данные в новый файл **no-users** и одновременно выводит их в стандартный вывод. После

этого мы используем команду `cat`, чтобы перепроверить содержимое нового файла:

```
packt@neptune:~$ wc -l /etc/passwd | tee no-users
34 /etc/passwd

packt@neptune:~$ cat no-users
34 /etc/passwd
```

Команда `tee` — своего рода «гадкий утенок» среди команд, которые манипулируют файлами. Хотя она довольно мощная, подходящие случаи для ее использования можно легко упустить из виду. Тем не менее мы рекомендуем обращаться к `tee` как можно чаще.

В следующем разделе мы рассмотрим, как использовать текстовые редакторы в командной строке Linux.

Как создавать и редактировать файлы с помощью текстовых редакторов

Linux предлагает широкий выбор текстовых редакторов с интерфейсом командной строки. Самые популярные из них — **nano**, **Emacs** и **Vim**, но стоит упомянуть также **Pico**, **JOE** и **ed**. Здесь мы начнем с Vim, потому что вы, вероятнее всего, встретитесь с ним в любой системе Linux, с которой будете работать. Впрочем, в последнее время в качестве текстового редактора по умолчанию все чаще используют nano. Например, Ubuntu по умолчанию поставляется без Vim¹, хотя в CentOS он включен, а Fedora стремится сделать nano стандартным текстовым редактором². Поэтому, возможно, вам стоит освоить nano, но, чтобы чувствовать себя уверенно в более консервативном окружении, имеет смысл овладеть редактором Vim.

Как редактировать текстовые файлы с помощью Vim

Vim — это улучшенная версия **vi**, текстового редактора по умолчанию в Unix. Это очень мощный инструмент со множеством функций, которые могут облегчить вам работу, хотя освоить их все не так-то просто. В этом разделе мы познакомим вас с основными командами, которых будет достаточно, чтобы эффективно использовать Vim.

¹ На момент подготовки этого издания (2025 г.) в дистрибутивы Ubuntu включена ограниченная версия Vim. Ее отличия от полнофункционального редактора Vim можно узнать с помощью команды `vim --version`. — *Примеч. ред.*

² В дистрибутивах Fedora nano служит текстовым редактором по умолчанию, начиная с версии 33 (октябрь 2020 г.). — *Примеч. ред.*

В Vim предусмотрено несколько режимов работы, между которыми можно переключаться. Вот основные режимы:

- *нормальный*, или *командный*, режим — это режим по умолчанию, в котором редактор ожидает команд;
- режим *вставки* текста;
- *визуальный* режим, который позволяет выделять фрагменты текста и выполнять операции над ними;
- режим *командной строки*.

Давайте посмотрим, как переключаться между этими режимами.

Как переключаться между режимами

Когда вы впервые открываете Vim, перед вами появляется пустое окно редактора, в котором отображаются только сведения о версии программы и перечень важнейших команд. Это значит, что вы находитесь в нормальном (командном) режиме, где Vim ожидает ваших команд.

Чтобы перейти в режим вставки, нажмите клавишу **I**, и вы сможете вставлять текст, начиная с текущей позиции курсора. Можно также нажать **A**, чтобы начать редактирование справа от позиции курсора. Чтобы выйти из режима вставки, нажмите **Esc**, и вы вернетесь в нормальный режим.

Если вы открыли файл, в котором уже есть текст, то в нормальном режиме по нему можно перемещаться с помощью клавиш со стрелками. Поскольку Vim унаследовал рабочий процесс **vi**, вместо стрелок можно также нажимать **H** (влево), **J** (вниз), **K** (вверх) и **L** (вправо). Эти клавиши остались с тех времен, когда на клавиатурах терминалов не было отдельных клавиш со стрелками.

Из нормального режима можно также переключиться в режим *замены*, нажав **R**. В этом режиме можно заменить символ, который находится непосредственно в позиции курсора.

В нормальном режиме можно инициировать поиск по тексту, нажав **/**. После этого введите шаблон для поиска и нажмите **Enter**.

Существует также режим *командной строки*, который активируется нажатием клавиши **:** (двоеточие). Это расширенный режим, в котором можно вводить такие команды, как **w** (сохранить файл), **q** (выйти из Vim) или **wq** (сохранить и выйти).

Основные команды Vim

В работе с Vim важно, чтобы вы уверенно владели сочетаниями клавиш для основных операций. Чтобы ознакомиться со всеми доступными командами, обратитесь к документации по Vim (vimhelp.org), а здесь мы вкратце перечислим самые полезные сочетания клавиш:

yy	Скопировать блок текста в буфер обмена
p	Вставить скопированный блок
u	Отменить последнюю операцию
x	Удалить символ справа от курсора
X	Удалить символ слева от курсора
dd	Удалить строку, на которой находится курсор
h	Влево
l	Вправо
k	Вверх
j	Вниз
w	Вправо, к началу следующего слова
b	Влево, к началу предыдущего слова
^	В начало строки
\$	В конец строки
gg	В начало документа
G	В конец документа
:n	Перейти к строке с номером <i>n</i>
i	Вставить перед позицией курсора
I	Вставить в начало строки
a	Вставить справа от курсора
A	Вставить в конец строки
O	Вставить в начало следующей строки
/	Искать строки по шаблону
?	Поиск назад от позиции курсора
n	Следующая позиция искомой строки
N	Предыдущая позиция искомой строки
:wq	Сохранить изменения и выйти из Vim
:q!	Выйти без сохранения
:w!	Сохранить изменения, но не выходить
ZZ	Сохранить и закрыть файл
:w <i>файл</i>	Сохранить как новый файл с именем <i>file</i>

Vim может отпугивать новичков в Linux, и вам нечего стесняться, если вы предпочитаете другие редакторы — ведь их существует великое множество. Сейчас мы познакомим вас с **nano**.

Текстовый редактор nano

Vim — мощный текстовый редактор, и системному администратору важно уметь им пользоваться. Однако есть и другие редакторы, которые не менее функциональны и при этом проще в использовании.

К ним относится **nano**, который по умолчанию установлен в Ubuntu и Rocky Linux и который можно запускать «из коробки» в обоих дистрибутивах. Редак-

тор по умолчанию не задается в файле `.bashrc` с помощью переменной `$EDITOR`, однако следующая команда в Ubuntu позволяет проверить, какой редактор установлен по умолчанию:

```
packt@neptune:~$ sudo update-alternatives --config editor
There are 4 choices for the alternative editor (providing /usr/bin/editor).
```

Selection	Path	Priority	Status

* 0	/bin/nano	40	auto mode
1	/bin/ed	-100	manual mode
2	/bin/nano	40	manual mode
3	/usr/bin/vim.basic	30	manual mode
4	/usr/bin/vim.tiny	15	manual mode

```
Press <enter> to keep the current choice[*], or type selection number: _
```

Редактор nano можно вызвать с помощью команды `nano` и в Ubuntu/Debian, и в Fedora/Rocky Linux, и в openSUSE. Интерфейс nano гораздо проще, чем, например, у Vim или Emacs. Используйте тот текстовый редактор, который вам больше нравится.

Итоги

В этой главе вы научились работать с самыми распространенными командами в Linux. Теперь вы знаете, как обращаться с файлами (создавать, удалять, копировать и перемещать их), как организована файловая система, как работать с каталогами и как просматривать содержимое файлов. У вас есть представление о том, что такое оболочка и как устроены основные права доступа. Полученные знания помогут вам управлять файлами в любом дистрибутиве Linux и редактировать текстовые файлы. Вы научились работать с Vim — одним из самых распространенных текстовых редакторов с интерфейсом командной строки в Linux. Эти навыки помогут вам овладеть другими текстовыми редакторами — такими как nano и Emacs. Ваши умения пригодятся почти в каждой главе этой книги, а также в повседневной работе в качестве системного администратора.

В следующей главе вы узнаете, как управлять пакетами, в том числе устанавливать, удалять и запрашивать пакеты в дистрибутивах на основе Debian и Red Hat. Эти навыки важны для любого администратора и входят в обязательную программу профессиональной подготовки.

Вопросы

Во второй главе мы рассмотрели файловую систему Linux и основные команды, на которые будет опираться весь дальнейший материал этой книги. Вот несколько вопросов, которые помогут вам проверить свои знания и попрактиковаться в командной строке:

1. Как создать сжатый архив со всеми файлами из каталога `/etc`, у которых есть расширение `.conf`?

Подсказка: используйте команду `tar` так, как показано в этой главе.

2. Как вывести список пяти самых больших файлов в каталоге `/etc` по порядку от большего к меньшему?

Подсказка: используйте `find` в сочетании с `sort` и `head`.

3. Как создать цепочку вложенных каталогов?

Подсказка: используйте `mkdir` так, как показано в этой главе.

4. Как найти в корневом каталоге файлы с тремя разными расширениями?

Подсказка: используйте команду `find`.

5. Для каких команд в Linux установлен бит SUID?

Подсказка: используйте команду `find` с ключом `-perm`.

6. Как создать файл с 1000 строк случайно сгенерированных слов (по одному слову в строке)?

Подсказка: используйте команду `shuf` (не рассматривалась в этой главе).

7. Как создать файл с 1000 случайно сгенерированных чисел?

Подсказка: используйте цикл `for`.

8. Как узнать, когда в последний раз использовался `sudo` и какие команды были выполнены с его помощью?

Подсказка: используйте команду `grep`.

Дополнительная литература

- Оливер Пельц (Oliver Pelz). «Fundamentals of Linux».
- Джей Лакруа (Jay LaCroix). «Mastering Ubuntu Server».

3

Как управлять программным обеспечением в Linux

https://t.me/it_boooks/2

Управление программным обеспечением — важная часть работы системного администратора Linux. Эта глава научит вас эффективно обращаться с программными пакетами.

Вы освоите команды, которые необходимы, чтобы управлять программным обеспечением, а также узнаете, как обращаться с пакетами ПО в разных дистрибутивах. Вы познакомитесь с новейшими типами пакетов **Snap** и **Flatpak** и научитесь использовать их в современных дистрибутивах Linux.

В этой главе мы рассмотрим такие темы:

- Какие бывают типы пакетов программного обеспечения в Linux.
- Как управлять пакетами программного обеспечения.
- Как устанавливать новые среды рабочего стола в Linux.

Технические требования

В этой главе нет никаких особых технических требований; достаточно, чтобы у вас был действующий экземпляр Linux. Для упражнений этой главы одинаково хорошо подходят **Ubuntu**, **Fedora** (или **AlmaLinux**) или **openSUSE**, потому что мы рассмотрим все разновидности систем управления пакетами.

Типы пакетов программного обеспечения в Linux

Как вы уже знаете, в состав любого дистрибутива Linux входит ядро и приложения, которые работают поверх него. Хотя многие из них уже установлены по умолчанию, время от времени вам непременно понадобится устанавливать новые приложения или удалять ненужные.

Приложения в Linux поступают из **репозитория**. Репозиторий — это централизованно управляемое хранилище пакетов программного обеспечения, которые поддерживают разработчики. В пакете могут содержаться отдельные приложе-

ния или дополнительные компоненты операционной системы. Каждый дистрибутив Linux поставляется в связке с несколькими официальными репозиториями, но помимо них можно вручную добавить и другие. Это делается по-разному в разных дистрибутивах, и далее в этой главе мы поговорим об этом подробнее.

В Linux существует несколько типов пакетов. Ubuntu, которая основана на Debian, использует пакеты в формате deb, а Fedora (так же, как Rocky Linux и AlmaLinux) основана на RHEL и использует пакеты RPM. Они же применяются в openSUSE, хотя изначально она была основана на Slackware. Кроме двух перечисленных типов, в последнее время завоевывают популярность относительно новые форматы пакетов: Snap, который разработала компания Canonical (она же выпускает Ubuntu), и Flatpak — этот формат создало большое сообщество разработчиков и компаний, среди которых GNOME, Red Hat и Endless.

Типы пакетов deb и RPM

deb и RPM — классические типы пакетов, которые используются соответственно в Ubuntu и Fedora. Они по-прежнему широко распространены, хотя новые типы (Snap и Flatpak) начинают завоевывать позиции в настольных версиях Linux.

Оба типа пакетов соответствуют стандарту **Linux Standard Base (LSB)**¹, последняя версия которого (LSB 5.0) выпущена в 2015 году. Дополнительную информацию о ней можно найти по адресу refspecs.linuxfoundation.org/lsb.shtml#PACKAGEFMT.

Как устроен пакет deb

Формат пакетов deb был представлен в дистрибутиве Debian в 1993 году и с тех пор используется во всех дистрибутивах, производных от Debian и Ubuntu. Пакет deb — двоичный (бинарный): это значит, что он содержит файлы самой программы, а также ее зависимости и файлы с метаданными, причем все эти ресурсы упакованы в архив.

Чтобы проверить содержимое пакета deb, можно использовать команду `ar`. Она не установлена по умолчанию в Ubuntu 22.04.2 LTS, поэтому вам придется установить ее самостоятельно с помощью такой команды:

```
$ sudo apt install binutils
```

Вот и все — вы установили пакет в Ubuntu! Теперь с помощью `ar` можно проверить содержимое любого пакета deb. Давайте в качестве упражнения загрузим

¹ После выпуска LSB 5.0 этот стандарт перестал развиваться и сопровождаться, и современные системы семейства Linux не гарантируют его поддержку. Debian прекратил поддерживать LSB в сентябре 2015-го, RHEL — в 2019 году. — *Примеч. ред.*

пакет менеджера паролей **1password** и проверим его содержимое. Чтобы изучить пакет, выполните следующие действия:

1. Запустите команду `wget`, чтобы загрузить файл пакета в ваш текущий рабочий каталог:

```
wget https://downloads.1password.com/linux/debian/amd64/
stable/1password-latest.deb
```

2. После этого воспользуйтесь командой `ar t 1password-latest.deb`, чтобы просмотреть содержимое двоичного пакета. Ключ `t` позволяет вывести оглавление архива:

```
packt@neptune:~$ ar t 1password-latest.deb
debian-binary
control.tar.xz
data.tar.xz
_gpgorigin
```

Как видите, в выводе перечислены четыре файла, два из которых являются архивами. С помощью команды `ar` можно изучать пакет еще глубже.

3. Чтобы извлечь содержимое пакета в текущий рабочий каталог, запустите такую команду:

```
$ ar x 1password-latest.deb
```

4. С помощью команды `ls` просмотрите содержимое текущего каталога. Вы увидите, что четыре файла распакованы и готовы к просмотру. `debian-binary` — это текстовый файл, который содержит версию формата файла пакета — в нашем случае это 2.0. Просмотрите этот файл с помощью такой команды:

```
$ cat debian-binary
2.0
```

5. В архиве `control.tar.gz` находятся файлы с метаданными и сценарии, которые будут запускаться во время установки или до и после нее, в зависимости от ситуации. Архив `data.tar.xz` содержит исполняемые файлы и библиотеки программы, которые извлекаются во время установки. Следующая команда позволяет просмотреть содержимое этого архива:

```
$ tar tJf data.tar.xz | head
./
./opt/
./opt/1Password/
./opt/1Password/1Password-BrowserSupport
./opt/1Password/1Password-Crash-Handler
./opt/1Password/1Password-LastPass-Exporter
./opt/1Password/1password
./opt/1Password/LICENSE.electron.txt
./opt/1Password/LICENSES.chromium.html
./opt/1Password/after-install.sh
```

6. Последний файл — это файл подписи `gpg`.

В файле `gpg` используется шифрование с помощью программы GnuPG (GNU Privacy Guard), которая использует стандарт шифрования OpenPGP (определенный спецификацией RFC 4880¹). Обычно файлы пакетов подписываются с его помощью, что позволяет разработчикам безопасно распространять программное обеспечение. Чтобы узнать подробности, можно ознакомиться с официальной документацией: openpgp.org.

На приведенном ниже скриншоте показан вывод этих команд.

```
packt@neptune:~$ ls
lpassword-latest.deb  development  _gpgorigin  package-list
backup_dir1          dir1         links        poem
control.tar.gz       files        locate-search poem-spaces
data.tar.xz          files-archive-gzipped.tar.gz new-report   reports
debian-binary        files-archive.tar no-users     users
packt@neptune:~$ cat debian-binary
2.0
packt@neptune:~$ tar tJf data.tar.xz | head
./
./usr/
./usr/share/
./usr/share/doc/
./usr/share/doc/lpassword/
./usr/share/doc/lpassword/changelog.gz
./usr/share/icons/
./usr/share/icons/hicolor/
./usr/share/icons/hicolor/128x128/
./usr/share/icons/hicolor/128x128/apps/
```

Рис. 3.1. Содержимое пакета deb

Метаданные каждого пакета — это набор файлов, без которых программа не сможет работать. Они содержат информацию о требованиях пакета, а также обо всех его зависимостях, конфликтах и предпочтениях. Чтобы подробно изучить, из чего состоит пакет, достаточно использовать только команды, связанные с пакетами.

Теперь, когда мы знаем, как устроен пакет в формате Debian, давайте рассмотрим компоненты пакета Red Hat.

¹ В июле 2024 года IETF выпустил стандарт RFC 9580 (rfc-editor.org/rfc/rfc9580), который заменяет собой RFC 4880 и не обеспечивает обратную совместимость с ним. Разработчики GnuPG не поддержали новый RFC и используют собственную конкурирующую спецификацию LibrePGP, которая опирается на RFC 4880 (datatracker.ietf.org/doc/draft-koch-librepgp). — *Примеч. ред.*

Как устроен пакет RPM

Пакеты **RPM (RPM Package Manager¹)** разработаны компанией Red Hat и используются в Fedora, CentOS, RHEL, AlmaLinux, Rocky Linux, SUSE и openSUSE. Двоичные пакеты RPM похожи на deb тем, что они тоже упакованы в архив.

Давайте проверим пакет **1password** в формате RPM подобно тому, как мы делали это с пакетом deb в предыдущем разделе:

1. Загрузите пакет RPM с помощью такой команды:

```
# wget https://downloads.1password.com/linux/rpm/stable/  
x86_64/1password-latest.rpm
```

Если вы попытаетесь использовать ту же команду **ar**, то убедитесь, что в случае с RPM она не распознает формат файла. Однако здесь на помощь приходят другие, более мощные инструменты.

2. Мы будем использовать команду **rpm** — специальное низкоуровневое средство управления пакетами RPM — с ключами **-q** (запросить данные), **-p** (имя пакета) и **-l** (вывести список файлов в пакете):

```
# rpm -qp1 1password-latest.rpm
```

В результате, в отличие от пакета deb, вы увидите список всех файлов, которые относятся к приложению, а также сведения о том, куда каждый из них будет установлен в вашей системе.

3. Чтобы просмотреть метаданные пакета, запустите команду **rpm** с ключами **-q**, **-p** и **-i** (отобразить метаданные). Вот что выводит команда:

```
[packt@fedora ~]$ rpm -qpi 1password-latest.rpm  
warning: 1password-latest.rpm: Header V4 RSA/SHA512 Signature,  
key ID 2012ea22: NOKEY  
  
Name       : 1password  
Version    : 8.10.58  
Release    : 1  
Architecture: x86_64  
Install Date: (not installed)  
Group      : default  
Size       : 613123824  
License    : LicenseRef-1Password-Proprietary  
Signature  : RSA/SHA512, Tue Jan 14 22:23:13 2025, Key ID ac2d62742012ea22  
Source RPM : 1password-8.10.58-1.src.rpm  
Build Date : Tue Jan 14 22:18:58 2025  
Build Host : runner-wuw4zwa3x-project-152-concurrent-9-oizrkxiz  
Relocations : /  
Packager   : 1Password <support@1password.com>  
Vendor     : 1Password <support@1password.com>  
URL        : https://1password.com
```

¹ Изначально аббревиатура RPM расшифровывалась как Red Hat Package Manager, но с 2000 года официально считается рекурсивным акронимом от RPM Package Manager. — *Примеч. ред.*

```
Summary      : Password manager and secure wallet
Description  :
Password manager and secure wallet
```

Здесь выводятся сведения о приложении: его название, версия, номер выпуска, архитектура, дата установки, группа, размер, лицензия, подпись, исходный файл RPM, дата и узел сборки, официальный сайт, возможность установки в другой каталог и описание.

4. Чтобы узнать, какие еще зависимости потребуются пакету при установке, запустите ту же команду `rpm` с ключами `-q`, `-p` и `--requires`:

```
$ rpm -qp --requires 1password-latest.rpm
warning: 1password-latest.rpm: Header V4 RSA/SHA512 Signature, key ID
2012ea22: NOKEY
/bin/sh
/bin/sh
libasound.so.2()(64bit)
libatk-1.0.so.0()(64bit)
libatk-bridge-2.0.so.0()(64bit)
libdrm.so.2()(64bit)
libgbm.so.1()(64bit)
libgdk_pixbuf-2.0.so.0()(64bit)
libgtk-3.so.0()(64bit)
libnss3.so()(64bit)
libuuid.so.1()(64bit)
libxshmfence.so.1()(64bit)
rpmlib(CompressedFileNames) <= 3.0.4-1
rpmlib(PayloadFilesHavePrefix) <= 4.0-1
rpmlib(PayloadIsXz) <= 5.2-1
udev
xdg-utils
```

Теперь вы знаете, как устроены пакеты Debian и Red Hat и чем они различаются. Имейте в виду, что, помимо этих двух типов, в разных дистрибутивах Linux доступны и другие форматы пакетов. Кроме того, как уже упоминалось, существуют новые кроссплатформенные типы пакетов для Linux, которые называются Flatpak и Snap и о которых мы подробно расскажем в следующем разделе.

Типы пакетов Snap и Flatpak

Snap и **Flatpak** — относительно новые типы пакетов, и с ними связывают будущее приложений в Linux. Пакеты обоих типов собирают и запускают приложения в изолированных контейнерах, что делает их безопаснее и переносимее. И тот и другой формат пакетов были созданы, чтобы облегчить установку настольных приложений и улучшить их переносимость.

Хотя основные дистрибутивы Linux располагают крупными репозиториями приложений, для независимых поставщиков программного обеспечения или сопровождающих волонтеров может оказаться проблематичным распространять программное обеспечение для такого «зоопарка» дистрибутивов Linux, в каждом

из которых используются свои собственные типы пакетов. Именно здесь на помощь приходят Snap и Flatpak, которые призваны облегчить хлопоты, связанные с распространением программ.

Предположим, что вы — независимый разработчик программного продукта для Linux. Как только появится новая версия вашего продукта, вам понадобится создать как минимум два типа пакетов, которые можно будет загрузить напрямую с вашего сайта, — пакет `.deb` для Debian/Ubuntu/Mint и родственных систем и пакет `.rpm` для Fedora/RHEL/SUSE и других дистрибутивов этого семейства.

Но если вы хотите избежать этого раздвоения и сделать так, чтобы приложение было доступно в виде универсального пакета для большинства существующих дистрибутивов Linux, его можно распространять в формате Flatpak или Snap. Пакет Flatpak будет доступен через централизованный репозиторий Flathub, а пакет Snap — через репозиторий Snap Store. И то и другое одинаково хорошо подходит для нашей цели: распространять приложение для всех основных дистрибутивов Linux с минимальными затратами ресурсов и централизованными усилиями.

И Snap, и Flatpak пытаются преодолеть одну и ту же проблему: глобальную фрагментацию экосистемы Linux в той части, которая относится к пакетам. Однако эти два формата опираются на разные предпосылки. Snap изначально предназначался для интернета вещей и серверных версий Ubuntu от Canonical, в то время как Flatpak создавался как универсальный тип пакета для настольных приложений в Linux. В результате пакеты Flatpak доступны только в настольных системах, а не в серверных или встраиваемых версиях Linux. По мере развития обоих форматов все больше дистрибутивов начинают предоставлять их по умолчанию, причем Flatpak лидирует по этому показателю. Пакеты Snap по умолчанию доступны прежде всего в официальных выпусках Ubuntu, начиная с версии 23.04, а пакеты `jSnap` — в Fedora, openSUSE, Pop!_OS, Linux Mint, KDE neon и других дистрибутивах.

Из всего этого можно сделать такой вывод: для того чтобы распространять программное обеспечение для Linux, приходится прилагать больше усилий, чем в случае со сборкой того же приложения для Windows или macOS. Хотелось бы надеяться, что в будущем останется лишь один универсальный формат пакетов для распространения программ под Linux, и от этого будет лучше и пользователям, и разработчикам.

Как устроен пакет Snap

Пакет Snap — это файл в формате **SquashFS**. Это значит, что он содержит собственную файловую систему, которая инкапсулирована в неизменяемый контейнер. У него очень строгое окружение, с особыми ограничениями и правилами изоляции. В каждом файле Snap есть каталог с метаданными, где хранятся файлы, которые управляют поведением пакета.

Snap, в отличие от Flatpak, используется не только для настольных приложений, но и для более широкого спектра серверных и встраиваемых приложений. Это связано с тем, что Snap берет свое начало от дистрибутива Ubuntu Snappy, который создавался для интернета вещей и смартфонов и озаменовал собой важное достижение компании Canonical (которая создала Ubuntu) на пути к технологической согласованности.

Как устроен пакет Flatpak

Формат Flatpak основан на технологии, которая прежде называлась **OSTree**; ее создали разработчики из Red Hat и сообщества GNOME, и сейчас она активно используется в Fedora Silverblue как основа для **rpm-ostree**. Это новая система обновления для Linux, которая предназначена для того, чтобы работать вместе с существующими системами управления пакетами. Она была вдохновлена технологиями Git и функционирует похожим образом: фактически это система контроля версий на уровне ОС. Она использует хранилище объектов с адресацией по содержимому, позволяет совместно обращаться к веткам и предлагает транзакционные обновления, а также откат и мгновенные снимки для ОС.

Некоторое время назад проект сменил название на **libostree**, чтобы подчеркнуть приоритет на проектах, которые уже используют одноименную библиотеку. Среди множества таких проектов мы рассмотрим только два: Flatpak и rpm-ostree. Последний считается гибридной системой управления пакетами нового поколения для таких дистрибутивов, как Fedora и CentOS/RHEL. Специальные версии этих дистрибутивов были основаны на проекте Atomic¹ — разработке команд Fedora и Red Hat, — который поддерживал неизменяемую инфраструктуру как для серверов, так и для настольных компьютеров. Разработчики openSUSE создали похожий инструмент Snapper, который создавал моментальные снимки ОС для файловой системы btrfs.

Flatpak использует инструменты libostree, которые похожи на rpm-ostree, но применяются только для контейнеризации настольных приложений и не управляют загрузчиком. Во Flatpak есть песочница, которая основана на другом проекте — **Bubblewrap**; он позволяет непривилегированным пользователям получать доступ к пространствам имен пользователей и обращаться к функциям контейнеров.

И пакеты Snap, и пакеты Flatpak можно устанавливать с помощью графического интерфейса, но можно и с помощью команд оболочки. В следующих разделах мы сосредоточимся исключительно на том, как работать со всеми типами пакетов в командной строке.

¹ В 2020 году проект Atomic Host был закрыт. Вместо дистрибутивов, которые опирались на эту технологию, сейчас выпускаются Fedora CoreOS и RHEL CoreOS: несмотря на похожие названия, контейнерная атомарность в них обеспечивается разными средствами. — *Примеч. ред.*

Как управлять пакетами программного обеспечения

В каждом дистрибутиве есть своя собственная **система управления пакетами**. Такие системы бывают двух типов: одни для низкоуровневого, а другие для высокоуровневого управления пакетами. В дистрибутивах на основе RPM, таких как CentOS или Fedora, низкоуровневым инструментом служит команда `rpm`, а высокоуровневыми — `yum` и `dnf`. В openSUSE — другом крупном дистрибутиве на основе RPM — на низком уровне используется та же команда `rpm`, а на высоком — `zypper`. В дистрибутивах на базе deb низкоуровневая команда — `dpkg`, а высокоуровневая — `apt` (или устаревшая `apt-get`).

В чем разница между низкоуровневыми и высокоуровневыми системами управления пакетами в Linux? Низкоуровневые системы манипулируют непосредственно самими пакетами и способны распаковывать пакеты, запускать сценарии и устанавливать приложения. В свою очередь, высокоуровневые системы умеют разрешать зависимости, устанавливать и загружать пакеты (и группы пакетов), а также искать метаданные.

Как управлять пакетами deb

Обычно в любом дистрибутиве пакетами управляет администратор или пользователь с правами `root` (`sudo`). Управлять пакетами — значит выполнять любые операции с ними: в частности, устанавливать, искать, загружать и удалять пакеты. Для всех этих операций существуют специальные команды Linux, и в следующих разделах мы разберемся, как их использовать.

Основные репозитории для Ubuntu и Debian

Официальные репозитории Ubuntu насчитывают около 60 000 пакетов, которые представлены в виде двоичных пакетов deb или пакетов Snap. Конфигурация системных репозиториев хранится в одном файле — `/etc/apt/sources.list`¹. В Ubuntu есть четыре основных репозитория, которые также называются ис-

¹ Начиная с версии Ubuntu 24.04, конфигурация репозиториев хранится в каталоге `/etc/apt/sources.list.d/`; в частности, системные репозитории перечислены в файле `/etc/apt/sources.list.d/ubuntu.sources`. Файлы `.sources` составляют в новом формате deb822, который считается более гибким и лучше структурированным, чем `.list`. В одной системе можно параллельно использовать файлы `.sources` и `.list`, если их содержание не конфликтует друг с другом. На момент подготовки этого издания (начало 2025 года) в Debian основная конфигурация репозиториев по-прежнему хранится в файле `/etc/apt/sources.list`, однако пользовательские источники пакетов рекомендуется размещать в отдельных файлах `.sources` в каталоге `/etc/apt/sources.list.d/`. Подробнее см. `man sources.list`. — *Примеч. ред.*

точниками пакетов, и они подробно описаны в файле `sources.list`. Эти репозитории таковы:

Main	Свободное программное обеспечение с открытым исходным кодом, которое поддерживает компания Canonical
Universe	Свободное программное обеспечение с открытым исходным кодом, которое поддерживается сообществом
Restricted	Проприетарное программное обеспечение
Multiverse ¹	Программное обеспечение, защищенное авторским правом

В файле `sources.list` по умолчанию активны все репозитории. Если вы хотите отключить какие-то из них, можете отредактировать файл соответствующим образом.

В Debian информация о репозиториях хранится в аналогичном файле `/etc/apt/sources.list`. Единственное отличие от Ubuntu состоит в том, что основные источники пакетов называются по-другому:

main	Программы, которые соответствуют стандартам Debian для свободного программного обеспечения
contrib	Программы, которые могут соответствовать или не соответствовать стандартам для свободного программного обеспечения, но не входят в основной дистрибутив
non-free	Программы, которые не распространяются с открытым исходным кодом и не соответствуют стандартам для свободного программного обеспечения

Конфигурационные файлы репозитория Debian и Ubuntu очень похожи, потому что записи в них организованы по одной и той же схеме. Различия проявляются в тех аспектах, которые специфичны для пакетов того или иного дистрибутива.

Работа с пакетами в обеих системах опирается на **APT** (Advanced Package Tool, «расширенный инструмент управления пакетами»), поэтому мы подробно рассмотрим его в следующем разделе.

Команды для пакетов APT

Несколько лет назад во всех дистрибутивах на базе Debian с пакетами можно было взаимодействовать с помощью команды `apt-get`, однако сейчас в качестве высокоуровневой системы управления пакетами используется новая команда `apt`,

¹ Main — основной, главный; Universe — вселенная; Restricted — закрытый, с ограниченным доступом; Multiverse — мультивселенная. — *Примеч. пер.*

которая лучше оптимизирована и структурирована, отчего работать с ней гораздо удобнее.

Прежде чем делать что-то полезное с помощью `apt`, нужно обновить список всех доступных пакетов в ваших репозиториях. Это можно сделать такой командой:

```
$ sudo apt update
```

В результате вы увидите, доступны ли какие-нибудь обновления и сколько пакетов следует обновить, а также какую команду можно запустить, чтобы получить более подробную информацию о них.

Прежде чем продолжить, давайте запустим команду `apt --help`, которая покажет самые востребованные команды, связанные с APT:

<code>list</code>	Перечислить пакеты по именам
<code>search</code>	Поиск по описанию пакетов
<code>show</code>	Подробные сведения о пакетах
<code>install</code>	Установить пакеты
<code>reinstall</code>	Переустановить пакеты
<code>remove</code>	Удалить пакеты
<code>autoremove</code>	Автоматически удалить все неиспользуемые пакеты
<code>update</code>	Обновить список доступных пакетов
<code>upgrade</code>	Установить и/или обновить пакеты
<code>full-upgrade</code>	Удалить, установить и/или обновить пакеты
<code>edit-sources</code>	Редактировать файл с данными об источниках пакетов
<code>satisfy</code>	Обработать строки зависимостей

Рис. 3.2. Наиболее часто используемые команды `apt`

Давайте рассмотрим некоторые из этих команд подробнее.

Как устанавливать и удалять пакеты

Установка и удаление пакетов входят в основные задачи системного администратора. В этом разделе мы рассмотрим, как устанавливать и удалять пакеты с помощью команды `apt`.

Чтобы установить новый пакет, можно запустить команду `apt install`: мы использовали ее в начале этой главы, когда рассказывали о том, как устроен пакет `deb`. Вы наверняка помните, что мы устанавливали команду `ar`, чтобы просматривать содержимое пакетов `.deb`. Для установки мы запускали такую команду:

```
$ sudo apt install binutils
```

Эта команда установила в систему несколько пакетов, и в одном из них была команда `ar`, которая нам нужна. Команда `apt` автоматически устанавливает и все необходимые зависимости.

Чтобы удалить пакет, можно воспользоваться командами `apt remove` или `apt purge`. Первая из них удаляет установленные пакеты и все их зависимости, которые установила команда `apt install`. Вторая команда делает то же самое, но при этом также удаляет все конфигурационные файлы, созданные приложениями.

В следующем примере мы удаляем приложения `binutils` с помощью `apt remove`:

```
$ sudo apt remove binutils
```

Эта команда выведет список пакетов, которые будут удалены, и удалит их из системы, но перед этим запросит подтверждение. Это хорошая мера безопасности, потому что она позволяет проверить, какие файлы будут удаляться. Если вы полностью доверяете команде, можно добавить в ее конце ключ `-y`, который автоматически подставляет ответ *Yes (Да)* на все вопросы команды.

Однако `apt remove` не удаляет конфигурационные файлы, которые относятся к удаленному приложению. Чтобы узнать, какие файлы остались в системе, можно запустить команду `find`. Например, чтобы узнать, какие файлы, связанные с пакетом `binutils`, не были удалены, можно вызвать такую команду:

```
sudo find / -type d -name *binutils 2>/dev/null1
```

В результате будут перечислены каталоги (потому что использовался ключ `-type d`), в которых после удаления пакета остались файлы, связанные с `binutils`.

Еще один инструмент, который позволяет удалять пакеты и все связанные с ними конфигурационные файлы, — `apt purge`. Эту команду можно использовать вместо `apt remove`:

```
$ sudo apt purge binutils
```

В результате вы увидите такие же сведения, как для `apt remove`: какие пакеты будут удалены и сколько места освободится на диске. Эта команда тоже запрашивает подтверждение, чтобы продолжить операцию.

Если вы собираетесь в качестве упражнения запустить `apt purge`, чтобы удалить тот же пакет (в нашем случае `binutils`), вам придется сначала установить его заново, потому что в предыдущем упражнении мы удалили его с помощью `apt remove`.

¹ Операнд `2>/dev/null` утилизирует стандартный поток ошибок, чтобы не загромождать вывод. Технически без него здесь можно обойтись. — *Примеч. ред.*

У команды `apt remove` также есть ключ `--purge`, который приводит к тому же результату, что и `apt purge`:

```
sudo apt remove --purge имя_пакета
```

Как уже говорилось, если удалять пакеты с помощью команды `apt remove`, то она оставляет нетронутыми некоторые конфигурационные файлы на случай, если вы удалили пакеты по ошибке и хотите восстановить предыдущую конфигурацию. Но если вы умышленно удалили пакет и теперь хотите избавиться от всех оставшихся файлов, то для этого можно запустить команду `apt purge`, передав ей имя того же удаленного пакета.

Как обновить систему

Время от времени имеет смысл обновлять всю систему, чтобы убедиться, что установлены все последние службы и обновления безопасности. В Ubuntu и Debian для этого традиционно применяется двухступенчатая процедура. Сначала мы с помощью `apt update` обновляем список репозиторий и убеждаемся, что в нем есть все сведения о доступных обновлениях. А затем мы запускаем `apt upgrade`, чтобы обновить пакеты. Оба этапа можно совместить в одной команде¹:

```
$ sudo apt update && sudo apt upgrade
```

Команда `update` иногда подсказывает, какие пакеты больше не нужны. Для этого она выводит сообщение такого вида:

```
The following packages were automatically installed and are no longer
required:
```

```
libfprint-2-tod1 liblvm9
```

```
Use 'sudo apt autoremove' to remove them2.
```

После того как вы обновили систему, можно запустить `sudo apt autoremove`, чтобы удалить ненужные пакеты. Эта команда, как и `remove`, покажет, какие пакеты будут удалены и сколько места освободится на диске, а также запросит подтверждение, чтобы продолжить операцию.

Теперь предположим, что вышел новый дистрибутив Ubuntu, и вы хотели бы перейти на него, потому что в нем есть улучшенные пакеты программного обеспечения, которое вас интересует. С помощью командной строки можно полностью обновить дистрибутив; соответствующая команда выглядит так:

```
$ sudo apt dist-upgrade
```

¹ Использование “&&” позволит выполнить `upgrade` только в случае успешного завершения `update`. — *Примеч. ред.*

² Следующие пакеты были установлены автоматически и больше не требуются: <...> Чтобы удалить их, запустите `sudo apt autoremove`. — *Примеч. пер.*

С той же целью можно запустить такую команду:

```
$ sudo apt full-upgrade
```

Обновление до новой версии дистрибутива обычно проходит гладко, но в отдельных случаях можно столкнуться с проблемами. Многое зависит от конфигурации, которую вы сами настроили. В любом случае мы советуем сделать полную резервную копию системы перед тем, как обновляться до новой версии.

Как работать со сведениями о пакетах

Работая с пакетами, иногда приходится прибегать к инструментам для сбора информации. Задачи системного администратора не ограничиваются тем, чтобы просто устанавливать и удалять пакеты; иногда нужно искать те или иные пакеты, изучать подробные сведения о них, создавать списки пакетов по определенным критериям и т. д.

Чтобы найти пакет, можно запустить команду `apt search`. Она выводит список всех пакетов, у которых в названии и других важных атрибутах есть искомая строка. Например, давайте поищем пакет `nmap`:

```
$ sudo apt search nmap
```

В результате вы увидите довольно длинный список пакетов, в свойствах которых так или иначе фигурирует строка `nmap`. Чтобы найти нужный пакет, этот список придется прокручивать вверх и вниз. Если вас интересуют более точные результаты, можно перенаправить вывод в команду `grep`, но при этом появится предупреждение, подобное тому, что показано на следующем снимке экрана:

```
packt@neptune:~$ sudo apt search nmap | grep nmap
```

WARNING: apt does not have a stable CLI interface. Use with caution in scripts¹.

```
IPv4 address parser for the nmap format
libcoq-mathcomp-finmap/noble 2.0.0-2build5 amd64
libnmap-parser-perl/noble 1.37-2 all
  module to parse nmap scan results with perl
librust-cpp-synmap-dev/noble 0.3.0-1 amd64
  Library for doing location lookup based on free openwlanmap.org data
  Library for doing location lookup based on free openwlanmap.org data
nmap/noble 7.94+git20230807.3be01efb1+dfsg-3build2 amd64
nmap-common/noble 7.94+git20230807.3be01efb1+dfsg-3build2 all
  Architecture independent files for nmap
nmapsi4/noble 0.5~alpha2-4build2 amd64
  graphical interface to nmap, the network scanner
python-libnmap-doc/noble 0.7.2-1 all
python3-libnmap/noble 0.7.2-1 all
```

¹ ПРЕДУПРЕЖДЕНИЕ: `apt` не обладает стабильным интерфейсом командной строки. Будьте осторожны, используя его в сценариях. — *Примеч. пер.*

```
python3-nmap/noble 0.6.1-1.1 all
  post-processor for TopHat unmapped reads
zenmap/noble 7.94+git20230807.3be01efb1+dfsg-3build2 all
```

После предупреждения вы видите короткий список пакетов, в названии или описании которых есть строка `nmap`, а среди них есть и сам искомый пакет `nmap`.

Чтобы избежать этого предупреждения, можно использовать устаревшую команду `apt-cache search`. Она тоже выводит список пакетов, но менее подробный, чем `apt search`:

```
packt@neptune:~$ sudo apt-cache search nmap | grep nmap
golang-github-malfunkt-iprange-dev - IPv4 address parser for the nmap format
libcoq-mathcomp-finmap - finite sets and maps extension for Mathematical
Components
libnmap-parser-perl - module to parse nmap scan results with perl
librust-cpp-synmap-dev - Sourcemap and full crate parsing support for `cpp_syn`
-
Rust source code
libwlocate-dev - Library for doing location lookup based on free openwlanmap.org
data
libwlocate0 - Library for doing location lookup based on free openwlanmap.org
data
nmap - The Network Mapper
nmap-common - Architecture independent files for nmap
nmapsi4 - graphical interface to nmap, the network scanner
python-libnmap-doc - Python NMAP Library (common documentation)
python3-libnmap - Python 3 NMAP library
python3-nmap - Python3 interface to the Nmap port scanner
tophat-recondition - post-processor for TopHat unmapped reads
zenmap - The Network Mapper Front End
```

Теперь, когда мы знаем, что в репозиториях Ubuntu есть пакет `nmap`, можно изучать его дальше, запросив более подробные сведения с помощью команды `apt show`:

```
$ apt show nmap
```

В результате вы увидите полную информацию о пакете, включая его название, версию, приоритет, происхождение и раздел, сведения о сопровождении, размер, зависимости, рекомендуемые дополнительные пакеты, размер загружаемых файлов, источники АРТ и описание.

В `apt` также есть полезная команда `list`: она выводит списки пакетов, которые удовлетворяют определенным критериям. Например, если просто запустить команду `apt list`, то она выведет список всех доступных пакетов. Но с помощью различных ключей можно сузить фильтр.

Чтобы показать установленные пакеты, используйте ключ `--installed`:

```
$ sudo apt list --installed
```

Давайте сформируем список всех доступных пакетов в одном файле, а всех установленных — в другом. Поскольку оба списка достаточно объемны, так их будет легче сравнить:

```
$ sudo apt list --installed > list-installed
$ sudo apt list > list
```

Можно сопоставить два полученных файла с помощью команды `ls -la` и посмотреть на разницу в размерах. Вы убедитесь, что файл `list` значительно больше, чем `list-installed`.

Сравнить эти два списка можно и другими способами, и мы предлагаем вам самостоятельно найти их в качестве упражнения. Смело используйте любые другие команды, связанные с APT, и побольше практикуйтесь, чтобы бегло владеть ими. APT — это мощный инструмент, и каждому системному администратору важно уметь им пользоваться, чтобы поддерживать работоспособность и исправность системы Linux. От того, какие приложения установлены и как они оптимизированы в масштабах всей системы, в значительной мере зависит удобство работы с ней.

Как управлять пакетами RPM

Пакеты в формате RPM используются в таких дистрибутивах Linux, как Fedora, AlmaLinux, Rocky Linux, RHEL и openSUSE/SLES. Для них существуют специальные инструменты высокого уровня, такие как `dnf`, `yum` и `zypper`, а низкоуровневым средством служит команда `rpm`.

В RHEL система управления пакетами по умолчанию называется YUM (Yellowdog Updater Modified¹), и в современных дистрибутивах это имя выступает просто аналогом DNF (Dandified YUM²) — аналогичной системы по умолчанию в Fedora. Если вы используете и Fedora, и RHEL, то для простоты можете смело пользоваться только одним из этих имен, потому что они вызывают одну и ту же команду. Во всех примерах в этой главе мы будем использовать YUM ради единообразия.

YUM — это стандартная система управления пакетами. Она способна устанавливать, удалять, обновлять и опрашивать пакеты, а также разрешать зависимо-

¹ Yellowdog — букв. Желтый Пес (*англ.*). Название системы происходит от дистрибутива Yellow Dog Linux, для которого был разработан YUP — предшественник YUM. Кроме того, `yum` (также `yumty`) — разговорное слово, которое обычно переводится как вкусно, аппетитный, ням-ням, вкусняшка (в прямом и переносном смысле). — *Примеч. пер.*

² Dandified — подчеркнуто стильный, лощеный, тщательно ухоженный (*англ.*). В названиях дистрибутивов, команд, пакетов, языков и других артефактов мира Unix и Linux используется много необычных слов, культурных отсылок, каламбуров и профессиональных мемов. — *Примеч. пер.*

сти. YUM работает с пакетами, которые установлены из репозитория, а также с локальными пакетами `.rpm`.

Основные репозитории для дистрибутивов на основе Fedora/RHEL

Для управления репозиториями служит каталог `/etc/yum.repos.d/`, а соответствующая конфигурация хранится в файле `/etc/dnf/dnf.conf`¹. Если вывести содержимое указанного каталога, результат будет примерно таким:

```
[packt@venus ~]$ ls -l /etc/yum.repos.d/
total 44
-rw-r--r--. 1 root root 1023 Nov 12 10:26 almalinux-appstream.repo
-rw-r--r--. 1 root root 987 Nov 12 10:26 almalinux-baseos.repo
-rw-r--r--. 1 root root 951 Nov 12 10:26 almalinux-crb.repo
-rw-r--r--. 1 root root 987 Nov 12 10:26 almalinux-extras.repo
-rw-r--r--. 1 root root 1107 Nov 12 10:26 almalinux-highavailability.repo
-rw-r--r--. 1 root root 951 Nov 12 10:26 almalinux-nfv.repo
-rw-r--r--. 1 root root 963 Nov 12 10:26 almalinux-plus.repo
-rw-r--r--. 1 root root 1107 Nov 12 10:26 almalinux-resilientstorage.repo
-rw-r--r--. 1 root root 939 Nov 12 10:26 almalinux-rt.repo
-rw-r--r--. 1 root root 999 Nov 12 10:26 almalinux-saphana.repo
-rw-r--r--. 1 root root 951 Nov 12 10:26 almalinux-sap.repo
```

Каждый из этих файлов содержит важную информацию о репозитории, такую как его название, список зеркал, расположение ключа `gpg` и статус эксплуатации. Все перечисленные файлы — это официальные репозитории.

Команды для пакетов YUM

С YUM связано множество команд и ключей, наиболее востребованные из которых позволяют устанавливать, удалять и искать пакеты, запрашивать сведения о них, обновлять систему и просматривать список репозитория.

Как устанавливать и удалять пакеты

Чтобы установить пакет из репозитория в AlmaLinux/Rocky Linux (или Fedora), просто запустите команду `yum install`. Например, так можно установить приложение GIMP из командной строки:

```
$ sudo yum install gimp
```

А если вы хотите установить пакет, который до этого загрузили локально, воспользуйтесь командой `yum localinstall`. Допустим, мы загрузили пакет `1password.rpm`:

```
wget https://downloads.1password.com/linux/rpm/stable/
x86_64/1password-latest.rpm
```

¹ В старых дистрибутивах — `/etc/yum.conf`. — *Примеч. ред.*

Его можно установить с помощью такой команды:

```
sudo yum localinstall 1password-latest.rpm
```

Команда `localinstall` автоматически разрешает необходимые зависимости и показывает источник (репозиторий) для каждой из них.

Это очень мощный инструмент, который в некоторых случаях позволяет вовсе обойтись без команды `rpm`. Команда `yum install` отличается от `yum localinstall` прежде всего тем, что последняя способна разрешать зависимости для локально загруженных пакетов. В то время как `install` ищет пакеты в активных репозиториях, `localinstall` ищет их в текущем рабочем каталоге.

Чтобы удалить пакет из системы, воспользуйтесь командой `yum remove`. Для примера удалим только что установленный пакет `1password`:

```
sudo yum remove 1password.x86_64
```

Система спросит, хотите ли вы удалить все пакеты, которые установило приложение. Выберите нужный вариант и продолжайте.

Если при таком запросе просто нажать `Enter` или `Return`, то в дистрибутивах, производных от Fedora или RHEL, ответом по умолчанию будет *N* (Нет), в то время как Ubuntu по умолчанию подставляет *Y* (Да). Это мера предосторожности, которая требует от вас повышенного внимания.

Команда `yum remove`, как и `yum install`, уведомляет о том, какие пакеты и зависимости будут удалены, если продолжить операцию.

Можете убедиться, что все зависимости, которые были установлены вместе с пакетом, удаляются с помощью `yum remove`. Если система предложит подтвердить действие, введите `y` и продолжите удаление.

Как обновить систему

Чтобы обновить систему на основе Fedora/RHEL, используйте команду `yum upgrade`. Есть также команда `yum update`, которая делает то же самое¹:

```
$ sudo yum upgrade
```

Можно использовать ключ `-y`, чтобы автоматически отвечать «да» на вопросы команды.

¹ Классическая команда `yum` вела себя немного по-разному с подкомандами `update` и `upgrade`, но в современных системах команды `yum update`, `yum upgrade` и `dnf upgrade` просто ссылаются на `dnf update`, а значит, все четыре команды действуют идентично. — *Примеч. ред.*

Наконец, существует также команда `upgrade-minimal`, которая устанавливает только последние обновления безопасности для пакетов.

Как работать со сведениями о пакетах

Управлять файлами с помощью `yum` можно почти так же, как с помощью `apt`. Для этого есть немало полезных команд, и сейчас мы подробно расскажем о тех, которые считаем наиболее распространенными. Чтобы узнать обо всех командах и о том, как их использовать, запустите `yum --help`.

Чтобы просмотреть историю команд `yum` и узнать, какие пакеты обрабатывались, запустите такую команду:

```
$ sudo yum history
```

В результате вы увидите отчет о каждой команде `yum`, которая запускалась ранее, в том числе количество измененных пакетов, а также время и дату выполнения действий:

```
[packt@venus ~]$ sudo yum history
```

ID	Command line	Date and time	Action(s)	Altered
4	upgrade -y	2025-02-02 15:05	I, U	97 <
3	remove 1password.x86_64	2025-02-02 13:34	Removed	1 >
2	localinstall 1password-1a	2025-02-02 13:24	Install	1
1		2025-02-02 01:55	Install	1190 EE

Команда `yum info` позволяет узнать подробности о том или ином пакете. Давайте опросим пакет `nmap` аналогично тому, как мы делали это в Ubuntu. В Rocky Linux команда будет выглядеть так:

```
[packt@venus ~]$ yum info nmap
```

```
Available Packages
Name       : nmap
Epoch     : 3
Version    : 7.92
Release    : 3.el9
Architecture : x86_64
Size       : 5.4 M
Source     : nmap-7.92-3.el9.src.rpm
Repository : appstream
Summary    : Network exploration tool and security scanner
URL        : http://nmap.org/
License    : Nmap
Description : Nmap is a utility for network exploration or security auditing.
            : It supports ping scanning (determine which hosts are up), many
            : port scanning techniques (determine what services the hosts are
            : offering), and TCP/IP fingerprinting (remote host operating
            : system identification).
            : ...
```

В результате вы увидите название, версию, выпуск, источник, репозиторий и описание — почти то же самое, что в случае с пакетами `.deb`.

Чтобы перечислить все доступные пакеты, можно вызвать команду `yum list`, а только установленные пакеты — `yum list installed`.

Если перенаправить вывод обеих команд в отдельные файлы, а затем сравнить их, мы увидим различия подобно тому, как мы делали это в Ubuntu. И та и другая команда выводит имя пакета, затем номер версии и выпуска, а также репозиторий, из которого он был установлен. Вот небольшой фрагмент:

```
xml-common.noarch           0.6.3-58.el9           @AppStream
xz.x86_64                   5.2.5-8.el9_0           @anaconda
xz-libs.x86_64              5.2.5-8.el9_0           @anaconda
yajl.x86_64                 2.1.0-22.el9            @AppStream
yum.noarch                  4.14.0-17.el9.alma.1    @anaconda
zip.x86_64                  3.0-35.el9              @anaconda
zlib.x86_64                 1.2.11-40.el9           @anaconda
```

Итак, мы рассмотрели наиболее востребованные команды для пакетов в форматах deb и RPM, но пока не рассказали о **Zypper** — системе управления пакетами для openSUSE и SUSE SLE. Далее мы вкратце рассмотрим ее основные команды, чтобы вы познакомились с Zypper и смогли увереннее осваивать openSUSE.

Как работать с Zypper

В openSUSE системой управления пакетами служит **Zypper**, который в целом работает так же, как `apt` в Debian/Ubuntu и `dnf` (или `yum`) в Fedora/RHEL. В следующих разделах мы рассмотрим некоторые его команды.

Как устанавливать и удалять пакеты

Подобно `apt` и `dnf`, система управления пакетами Zypper в openSUSE позволяет устанавливать и удалять пакеты с помощью почти такого же синтаксиса. Например, давайте используем команду `zypper`, чтобы установить `nmap`. Но сперва поищем это имя в соответствующих репозиториях, чтобы проверить, существует ли такой пакет. Для этого запустим такую команду:

```
sudo zypper search nmap
```

В результате мы увидим список пакетов, в имени которых есть строка `nmap`, с их типами и кратким описанием:

```
packt@localhost:~> sudo zypper search nmap
```

```
Loading repository data...
Reading installed packages...
```

S	Name	Summary	Type
	nmap	Network exploration tool and->	package
	nmap-parse-output	A tool for analyzing Nmap sc->	package
	nmap-parse-output-bash-completion	Bash Completion for nmap-par->	package

nmapsi4	A Graphical Front-End for Nmap	package
python3-chainmap	Backport/clone of ChainMap f->	package
python3-dragonmapper	Identification and conversio->	package
python311-dragonmapper	Identification and conversio->	package

Первый столбец списка озаглавлен буквой S. Это состояние пакета, и если пакет уже установлен, то в столбце будет соответствующее обозначение (например, i или v).

Из результатов поиска видно, что пакет для приложения Nmap называется `nmap` (он мог называться и по-другому, поэтому мы и использовали команду `search`). Чтобы установить его в системе, воспользуемся командой `zypper install`.

В openSUSE доступны короткие версии команд. Например, вместо `zypper install` можно ввести `zypper in`, после чего указать имя пакета, который вы хотите установить. Аналогично вместо `zypper update` (обновить пакет) можно использовать `zypper up`, а вместо `dist-upgrade` (обновить дистрибутив) — `dup`. Кроме того, команду `zypper remove` (удалить пакет) можно сократить как `zypper rm`. Дополнительную информацию вы найдете на справочных страницах.

Итак, вот команда для того, чтобы установить пакет `nmap`:

```
sudo zypper install nmap
```

С тем же успехом можно использовать такую команду:

```
sudo zypper in nmap
```

Результат показан на следующем снимке экрана:

```
packt@localhost:~> sudo zypper in nmap
Loading repository data...
Reading installed packages...
Resolving package dependencies...

The following 3 NEW packages are going to be installed:
  libpcap1 libssh2-1 nmap

3 new packages to install.
Overall download size: 5.6 MiB. Already cached: 0 B. After the operation,
additional 24.4 MiB will be used.
Continue? [y/n/v/ ... ? shows all options] (y):
```

Этот вывод показывает, какие пакеты будут установлены. Обратите внимание, что Зурпер, как и другие системы управления пакетами, автоматически обрабатывает зависимости. В данном случае, кроме `nmap`, нужно установить еще два библиотечных пакета. Введите `y`, чтобы продолжить операцию, и пакеты будут установлены.

Теперь давайте еще раз запустим `zypper search nmap` и посмотрим, как изменились результаты поиска:

S	Name	Summary	Type
i+	nmap	Network exploration tool and->	package
	nmap-parse-output	A tool for analyzing Nmap sc->	package
	nmap-parse-output-bash-completion	Bash Completion for nmap-par->	package
	...	...	...

На сей раз в первом столбце перед пакетом `nmap`, который мы только что установили, стоит `i+`. Это и означает, что пакет и его зависимости установлены. Таким образом, если вы ищете пакет, который уже установлен, то вы узнаете об этом из столбца состояния (первого столбца результатов поиска).

Теперь давайте удалим тот же самый пакет. Для этого воспользуемся такой командой:

```
sudo zypper remove nmap
```

Вместо `remove` можно использовать `rm`; результат в любом случае будет выглядеть так:

```
packt@localhost:~> sudo zypper rm nmap
Reading installed packages...
Resolving package dependencies...

The following package is going to be REMOVED:
  nmap

1 package to remove.

Package install size change:
      |          0 B    required by packages that will be installed
-24.7 MiB | -   24.7 MiB released by packages that will be removed

Backend: classic_rpmtrans
Continue? [y/n/v/...? shows all options] (y):
```

Эта команда сообщает, какие пакеты будут удалены. Как видите, удаляется только `nmap`, а другие зависимости, которые устанавливались вместе с ним, остаются в системе. Чтобы удалить их тоже, добавьте ключ `--clean-deps`:

```
packt@localhost:~> sudo zypper rm --clean-deps nmap
Reading installed packages...
Resolving package dependencies...

The following 3 packages are going to be REMOVED:
  libpcap1 libssh2-1 nmap

3 packages to remove.

Package install size change:
      |          0 B    required by packages that will be installed
-25.3 MiB | -   25.3 MiB released by packages that will be removed

Backend: classic_rpmtrans
Continue? [y/n/v/...? shows all options] (y):
```

Теперь, когда вы знаете, как устанавливать и удалять пакеты в openSUSE с помощью `zypper`, давайте посмотрим, как использовать этот инструмент, чтобы обновить всю систему.

Как обновить систему

Перед тем как обновлять систему, стоит посмотреть, какие обновления доступны. Для этого запустите такую команду:

```
zypper list-updates
```

В результате вы увидите все обновления, которые доступны в вашей системе. Чтобы установить их, выполните любую из таких команд:

```
sudo zypper update
sudo zypper up
```

Если вы используете эти команды без аргументов, как мы только что показали, то установятся все доступные обновления. Можно также обновить отдельные пакеты, если передать имя пакета в качестве аргумента команде `update`.

В openSUSE также есть полезные команды, которые позволяют настраивать блокировки пакетов, если вы не хотите, чтобы они обновлялись или удалялись. Давайте на примере того же пакета `nmap` посмотрим, как это сделать. Если вы уже удалили его, как мы продемонстрировали в предыдущем разделе, — пожалуйста, установите его снова. Далее мы заблокируем пакет, проверим блокировку, а затем разблокируем его обратно.

Чтобы заблокировать пакет, запустите команду `add-lock` или `zypper al`. Список заблокированных пакетов в системе выводит команда `zypper ll` (`list locks`), а чтобы снять блокировку с пакета, можно воспользоваться командой `zypper rl` (`remove locks`):

```
packt@localhost:~> sudo zypper al nmap
Specified lock has been successfully added.1
```

```
packt@localhost:~> sudo zypper ll
```

```
# | Name | Type | Repository | Comment
--+-+-----+-----+-----+-----
1 | nmap | package | (any)      |
```

```
packt@localhost:~> sudo zypper rl nmap
1 lock has been successfully removed.2
```

¹ Указанная блокировка успешно установлена. — *Примеч. пер.*

² 1 блокировка успешно снята. — *Примеч. пер.*

Теперь снова заблокируем пакет `nmap` и попробуем его удалить. Можете убедиться, что пакет не удаляется. Система сообщает о проблеме и предлагает выбрать вариант дальнейших действий:

```
packt@localhost:~> sudo zypper al nmap
Specified lock has been successfully added.

packt @localhost:~> sudo zypper rm nmap
Reading installed packages...
Resolving package dependencies...

Problem: 1: conflicting requests
Solution 1: do not ask to delete all solvables providing nmap.x86_64 =
7.94-lp156.1.2
Solution 2: remove lock to allow removal of nmap-7.94-lp156.1.2.x86_64

Choose from above solutions by number or cancel [1/2/c/d/?] (c):
```

Итак, вы убедились, что обновить openSUSE очень просто, а также узнали, как с помощью Zypper защищать пакеты блокировкой. В следующем разделе мы увидим, как получать сведения о тех или иных пакетах.

Как работать со сведениями о пакетах

Как и системы управления пакетами APT и dnf в Ubuntu и Fedora, Zypper в openSUSE позволяет запрашивать информацию о пакетах. Давайте снова вернемся к пакету `nmap` и получим дополнительные сведения о нем. Для этого запустим команду `zypper info`:

```
sudo zypper info nmap
```

Как видно на следующем скриншоте, мы получили почти такую же информацию, что и в дистрибутивах на основе Ubuntu и RHEL. Поскольку мы удалили пакет `nmap`, здесь указано, что пакет не установлен. Вывод заканчивается расширенным описанием, которое мы здесь не приводим:

```
Information for package nmap:
-----
Repository      : Non-OSS Repository
Name            : nmap
Version         : 7.94-lp156.1.2
Arch            : x86_64
Vendor          : openSUSE
Installed Size  : 24.7 MiB
Installed       : Yes
Status          : up-to-date
Source package  : nmap-7.94-lp156.1.2.src
Upstream URL    : https://nmap.org/
Summary         : Network exploration tool and security scanner
Description     : ...
```

Теперь давайте посмотрим, как управлять пакетами Flatpak и Snar в системах семейства Linux.

Как работать с пакетами Snap и Flatpak

Snap и Flatpak — это относительно новые типы пакетов, которые используются в различных дистрибутивах Linux. Этот раздел посвящен тому, как управлять этими пакетами. С пакетами Snap мы будем работать на примере Ubuntu, а с пакетами Flatpak — на примере Fedora, хотя, если приложить немного усилий, можно заставить оба типа пакетов функционировать в любом дистрибутиве.

Как работать с пакетами Snap в Ubuntu

Система управления пакетами Snap установлена по умолчанию в Ubuntu 22.04.2 LTS, поэтому вам не нужно ее специально устанавливать. Просто отыщите нужный пакет и установите его в систему. Чтобы продемонстрировать, как работать с пакетами Snap, мы будем использовать приложение Slack.

Поиск пакетов Snap

Slack можно установить из магазина приложений Snap Store. Чтобы убедиться, что это приложение доступно в магазине, его можно найти с помощью команды `snap find`:

```
$ snap find slack
```

В выводе команды вы увидите много пакетов, в названии или описании которых есть строка `slack` или которые связаны с приложением Slack, но только первый пакет в списке — тот, который мы ищем.

В любом дистрибутиве Linux могут сосуществовать одинаковые приложения, которые установлены из пакетов разных типов с помощью разных систем управления пакетами. Например, можно установить один экземпляр Slack из файла `deb` с официального сайта, а другой — из Snap Store.

Если команда `find` сообщает, что пакет доступен, можно приступить к его установке.

Как установить пакет Snap

Чтобы установить пакет Snap для приложения Slack, можно воспользоваться командой `snap install`:

```
packt@neptune:~$ sudo snap install slack
2025-02-03T16:12:43+03:00 INFO Waiting for automatic snapd restart...
slack 4.41.105 from Slack✓ installed
```

Теперь давайте посмотрим, как узнать дополнительные сведения о пакете Snap, который мы только что установили.

Как работать со сведениями о пакетах Snap

Если вы хотите узнать больше о пакете, можно воспользоваться командой `snap info`:

```
$ snap info slack
```

В результате вы увидите подробные сведения о пакете, в том числе его название, краткое и подробное описание, информацию об издателе и идентификатор, а ближе к концу — перечень доступных **каналов обновления**:

```
packt@neptune:~$ snap info slack
name:      slack
summary:   Team communication for the 21st century.
publisher: Slack✓
store-url: https://snapcraft.io/slack
contact:   https://get.slack.help/hc/en-us
license:   unset
description: ...
commands:
  - slack
snap-id:    JUJH91Ved74jd4ZgJCpzMBtYbPOzTlsD
tracking:   latest/stable
refresh-date: today at 16:12 MSK
channels:
  latest/stable:    4.41.105 2024-12-17 (178) 127MB -
  latest/candidate: ↑
  latest/beta:      4.42.115 2025-01-15 (181) 127MB -
  latest/edge:      ↑
installed:         4.41.105                (178) 127MB -
```

Каждому каналу соответствует своя версия приложения, и можно выбрать ту, которая вам нужна. По умолчанию команда `install` выбирает канал `stable`, но если вы хотите установить другую версию, добавьте во время установки ключ `--channel`. В предыдущем примере мы использовали канал по умолчанию.

Как просмотреть установленные пакеты Snap

Чтобы просмотреть список пакетов Snap, которые установлены в системе, можно запустить команду `snap list`. Хотя мы установили только Slack, вы увидите в списке и другие приложения. Некоторые из них — например, `core` и `snappy` — требуются системе и устанавливаются по умолчанию вместе с дистрибутивом:

```
packt@neptune:~$ snap list
```

Name	Version	Rev	Tracking	Publisher	Notes
bare	1.0	5	latest/stable	canonical✓	base
core18	20240920	2846	latest/stable	canonical✓	base
gnome-3-34-1804	0+git.3556cb3	93	latest/stable	canonical✓	-
gtk-common-themes	0.1-81-g442e511	1535	latest/stable	canonical✓	-
slack	4.41.105	178	latest/stable	slack✓	-
snappy	2.67	23545	latest/stable	canonical✓	snappy

Теперь мы узнаем, как обновлять пакеты Snap.

Как обновлять пакеты Snap

Пакеты Snap обновляются автоматически, поэтому вам не придется делать ничего вручную. Впрочем, если требуется, можно проверить наличие обновлений и установить их принудительно с помощью команды `snap refresh`, например:

```
$ sudo snap refresh slack
```

Если после этого вы хотите откатить приложение к предыдущей версии, можно запустить команду `snap revert`, например:

```
$ sudo snap revert slack
```

В следующем разделе мы рассмотрим, как включать и отключать пакеты Snap.

Как включать и отключать пакеты Snap

Если вы хотите временно не использовать приложение, его можно отключить с помощью команды `snap disable`. Чтобы вернуть приложение в эксплуатацию, запустите команду `snap enable`:

```
packt@neptune:~$ snap disable slack
error: access denied (try with sudo)1
packt@neptune:~$ sudo snap disable slack
slack disabled
packt@neptune:~$ sudo snap enable slack
slack enabled
```

Не забывайте использовать `sudo`, чтобы включить или отключить приложение, которое установлено из пакета Snap. Если же вам недостаточно просто отключить приложение, его можно полностью удалить.

Как удалить пакет Snap

Если удалить приложение, которое установлено из пакета Snap, вместе с ним удаляются соответствующие конфигурационные файлы, пользователи и данные. Для этого можно использовать команду `snap remove`, например:

```
$ sudo snap remove slack
```

После того как приложение удалено, его внутренние пользовательские, конфигурационные и системные данные хранятся в течение 31 дня. Соответствующие файлы называются снимками (snapshot). Они хранятся в заархивированном виде в каталоге `/var/lib/snapd/snapshots` и содержат следующие данные: файл `.json` с описанием снимка, файл `.tgz` с системными данными и специальные файлы `.tgz` с данными о пользователях каждой системы. Команда `ls` для упомянутого каталога покажет снимок, который был автоматически создан для Slack:

¹ Ошибка: в доступе отказано (попробуйте с помощью `sudo`). — *Примеч. пер.*

```
packt@neptune:~$ sudo snap remove slack
slack removed
```

```
packt@neptune:~$ sudo ls /var/lib/snapd/snapshots/
1_slack_4.41.105_178.zip
```

```
packt@neptune:~$ sudo unzip -l /var/lib/snapd/snapshots/1_slack_4.41.105_178.zip
Archive:  /var/lib/snapd/snapshots/1_slack_4.41.105_178.zip
```

Length	Date	Time	Name
15294	1980-00-00	00:00	archive.tgz
125	1980-00-00	00:00	user/root.tgz
453	1980-00-00	00:00	meta.json
97	1980-00-00	00:00	meta.sha3_384
15969			4 files

Если вы не хотите, чтобы снимки создавались, можно запустить команду `snap remove` с ключом `--purge`. Если приложение работает с большим объемом данных, его снимки могут быть весьма увесистыми и занимать дисковое пространство. Чтобы просмотреть снимки, которые сохранены в системе, используйте команду `snap saved`, например:

```
packt@neptune:~$ sudo snap saved
Set Snap Age Version Rev Size Notes
1 slack 18.8m 4.41.105 178 15.4kB auto
```

В нашем случае вывод показывает, что было удалено только одно приложение, а в первом столбце (`Set`) указан идентификатор соответствующего снимка. Если вам нужно удалить снимок, запустите команду `snap forget`. Например, так можно удалить снимок приложения Slack:

```
packt@neptune:~$ sudo snap forget 1
Snapshot #1 forgotten.1
packt@neptune:~$ sudo snap saved
No snapshots found.2
```

Чтобы убедиться, что снимок был удален, мы снова использовали команду `snap saved`, как показано в этом примере.

Пакеты Snap — универсальные и простые в использовании. Эту технологию продвигают разработчики Ubuntu, но в других дистрибутивах пакеты Snap используются существенно реже. Если вы хотите установить эти пакеты где-то еще, сначала установите `snapd` в соответствии с инструкциями на snapcraft.io/docs/installing-snapd и убедитесь, что он работает так, как нужно.

Теперь познакомимся с другим новичком в мире пакетов — Flatpak. Мы будем практиковаться на примере Fedora, но имейте в виду, что Flatpak также под-

¹ Снимок № 1 забыт. — *Примеч. пер.*

² Снимки не обнаружены. — *Примеч. пер.*

держивают такие дистрибутивы на основе Ubuntu, как Linux Mint и elementary OS, и такие дистрибутивы на основе Debian, как PureOS и Endless OS. Список всех поддерживаемых дистрибутивов можно найти на flatpak.org/setup.

Как работать с пакетами Flatpak в Fedora Linux

Поскольку пакеты Flatpak доступны только в виде настольных приложений, мы будем использовать Fedora Linux Workstation.

Подобно пакетам Snap, пакеты Flatpack — это изолированные приложения, которые запускаются в песочнице. Каждый пакет содержит все компоненты среды выполнения и библиотеки, которые нужны для работы. Flatpack поддерживает инструменты управления с графическим интерфейсом, а также полный набор команд, которые можно использовать из командной строки. Основная команда — **flatpak**, которая употребляется с различными подкомандами в зависимости от того, какую операцию с пакетами нужно выполнить. Чтобы увидеть их полный список, используйте такую команду:

```
$ flatpak --help
```

В следующих разделах мы подробно расскажем о некоторых широко используемых командах для управления пакетами Flatpak. Но перед этим — несколько слов о том, как идентифицируются приложения Flatpak и как они будут представлены в командной строке, чтобы вы не запутались в этом вопросе.

У каждого приложения есть идентификатор в формате *домен.разработчик.Приложение*, например `com.obsproject.Studio`. Здесь *домен* — это домен верхнего уровня (например, `com` или `org`), *Приложение* — название приложения, а *разработчик* — название компании, проекта, рабочей группы или другой организации или персоны, которая разработала приложение. Эта схема позволяет каждому разработчику легко публиковать и поставлять несколько приложений.

Как добавлять репозитории Flatpak

Если вы собираетесь устанавливать приложения, нужно настроить репозитории. В контексте Flatpak репозиторий также называется **удаленным расположением** (remote).

В нашей системе Fedora 37 уже установлен Flatpak, но нужно добавить репозиторий flathub. Это можно сделать с помощью команды `flatpak remote-add`:

```
$ sudo flatpak remote-add --if-not-exists flathub https://dl.flathub.org/repo/flathub.flatpakrepo
```

Здесь мы использовали ключ `--if-not-exists`, который останавливает команду и не сообщает об ошибках, если репозиторий уже существует. После того как вы добавили репозиторий, можно устанавливать пакеты из него, но сначала понадобится перезагрузить систему.

В Fedora 37 и предыдущих версиях не все приложения из репозитория Flathub были доступны по умолчанию, но, начиная с версии 38, разработчики стремятся предоставлять полный ассортимент приложений. Давайте посмотрим, как установить приложение из Flathub на рабочую станцию Fedora.

Как установить приложение Flatpak

Чтобы установить пакет, нужно знать, как он называется. Нужные приложения и названия соответствующих пакетов можно найти на сайте flathub.org. Давайте найдем приложение **OBS Studio**. Чтобы его установить, можно либо нажать на кнопку **Install (Установить)** в правом верхнем углу, либо воспользоваться командами, которые приводятся на этой же странице. Мы будем использовать такую команду:

```
$ sudo flatpak install flathub com.obsproject.Studio
```

В последних версиях Flatpak (начиная с 1.2) работает более простая команда:

```
$ sudo flatpak install com.obsproject.Studio
```

Обе команды приводят к одному и тому же результату: устанавливают приложение OBS Studio.

Как управлять приложениями Flatpak

После того как вы установили приложение, его можно запустить из командной строки таким образом¹:

```
$ flatpak run com.obsproject.Studio
```

Если вы хотите обновить все приложения и компоненты среды выполнения, запустите такую команду:

```
$ sudo flatpak update
```

Чтобы удалить пакет Flatpak, используйте команду `flatpak uninstall`:

```
$ sudo flatpak uninstall com.obsproject.Studio
```

Команда `flatpak list` выводит список всех установленных приложений и компонентов среды выполнения Flatpak:

```
[packt@fedora ~]$ flatpak list
Name                Application ID Version      Branch  Installation
OBS Studio          com.obsproject.Studio  31.0.1    stable  system
```

¹ В системах с графической средой рабочего стола (таких как Fedora Workstation) установленное приложение также можно запустить другими способами: например, щелкнуть мышью по соответствующему значку в меню приложений или начать вводить название в поле поиска и нажать Enter, как только в результатах поиска отобразится нужное приложение. — *Примеч. ред.*

Slack	com.slack.Slack	4.41.105	stable	system
Freedesktop Pla...	org.freedesktop.Platform	freedesktop...	24.08	system
Mesa	...top.Platform.GL.default	24.3.1	23.08	system
Mesa (Extra)	...top.Platform.GL.default	24.3.1	23.08...	system
Mesa	...top.Platform.GL.default	24.3.1	24.08	system
Mesa (Extra)	...top.Platform.GL.default	24.3.1	24.08...	system
openh264	...sktop.Platform.openh264	2.1.0	2.2.0	system
openh264	...sktop.Platform.openh264	2.4.1	2.4.1	system
Adwaita theme	org.kde.KStyle.Adwaita		6.6	system
KDE Application...	org.kde.Platform		6.6	system

Чтобы увидеть только установленные приложения, используйте ключ `--app`:

```
$ flatpak list --app
```

Мы перечислили самые распространенные команды, с помощью которых можно управлять пакетами Flatpak. Разумеется, существует и много других команд, которые мы здесь не рассматриваем; вы можете ознакомиться с ними в документации и проверить на своей системе. Справочник по командам Flatpak доступен на странице docs.flatpak.org/en/latest/flatpak-command-reference.html.

Пакеты Flatpak обеспечивают гибкость и предоставляют доступ к новейшим версиям приложений. Это полезно, если вы планируете использовать надежную базовую операционную систему, но ее недостатком является то, что по умолчанию с ней поставляются старые версии нужных приложений. Flatpak позволяет решить эту проблему и получить доступ к новым версиям. Просматривайте приложения, которые доступны на Flathub, и пробуйте те, которые покажутся вам интересными и полезными.

Теперь вы знаете, как устанавливать новые приложения в операционную систему с помощью командной строки. Но Linux также позволяет устанавливать новые среды рабочего стола. В следующем разделе мы покажем, как это сделать.

Как устанавливать среды рабочего стола в Linux

В качестве примера мы продолжим использовать Fedora, но команды из этого раздела можно применять и в любых других дистрибутивах на базе RHEL, например AlmaLinux или Rocky Linux.

По умолчанию в Fedora Workstation используется среда рабочего стола GNOME, но что, если вы хотите использовать другую — например, KDE? Прежде чем показать, как этого добиться, мы обзорно рассмотрим графические среды рабочего стола, которые доступны для Linux.

В мире Linux ценится возможность выбора, и это особенно наглядно проявляется, когда идет речь о **средах рабочего стола**. Существуют десятки сред — такие, как GNOME, KDE, Xfce, LXDE, LXQt, Pantheon и др. В Linux наиболее широко распространены GNOME, KDE и Xfce, причем первые две могут похвастаться самыми большими сообществами. Если вы хотите использовать

самые свежие версии GNOME, попробуйте такие дистрибутивы, как Fedora, openSUSE Tumbleweed (с GNOME) или Arch Linux/Manjaro. А если вас больше интересует KDE, обратите внимание на KDE neon или те же openSUSE Tumbleweed и Arch Linux/Manjaro с KDE. Что касается Xfce, то вы можете поработать с MX Linux, который основан на Debian и в котором по умолчанию установлена Xfce, или переключиться на Xfce в openSUSE. Как правило, наиболее распространенные дистрибутивы Linux предлагают модификации с разными средами рабочего стола; для Ubuntu такие модификации называются «сортами» (flavors), а для Fedora — «скрутками» (spins). Коммерческие версии RHEL и SUSE по умолчанию поставляются только с GNOME. Подробнее узнать о перечисленных средах рабочего стола можно на официальных сайтах:

KDE	kde.org
GNOME	gnome.org
Xfce	xfce.org

Теперь давайте посмотрим, как установить другую среду рабочего стола на Fedora Workstation.

Как установить KDE Plasma на Fedora Linux

В Fedora и производных дистрибутивах (а также в openSUSE) есть такой механизм, как **группы приложений**: он помогает устанавливать большие приложения и их зависимости. Это особенно полезно, когда вы собираетесь установить много приложений, составляющих большую *группу*, — это как раз относится к средам рабочего стола.

Чтобы установить группу¹, можно запустить команду `dnf install` и передать ей имя группы после знака `@`. Или используйте команду `dnf groupinstall`, заключив имя группы в кавычки.

¹ В книге описана установка KDE с помощью системы управления пакетами DNF 3. В Fedora Linux, начиная с версии 41, по умолчанию запускается DNF 5, которая во многом отличается от предыдущих версий. В частности, в DNF 5 применяется двух-уровневая иерархия: пакеты объединяются в *группы* (groups), а группы — в *окружения* (environments), причем KDE Plasma Workspaces — это окружение, а не группа.

Чтобы просмотреть список окружений, которые относятся к KDE, можно запустить такую команду DNF 5:

```
$ dnf environment list | grep KDE
```

Уточнить идентификатор нужного окружения и другие сведения о нем позволяет такая команда:

```
$ dnf environment info "KDE Plasma Workspaces"
```

Впрочем, установить KDE с помощью DNF 5 можно той же командой `sudo dnf install @kde-desktop-environment`, что приведена в книге.

Следующая команда позволяет проверить, какие группы из репозитория Fedora относятся к KDE:

```
$ dnf group list --all | grep "KDE"
```

В результате вы увидите список групп из репозитория Fedora, и среди них будет **KDE Plasma Workspaces**. Его можно установить с помощью любой из двух команд:

```
$ sudo dnf groupinstall "KDE Plasma Workspaces"
$ sudo dnf install @kde-desktop-environment
```

Так выглядит установка KDE Plasma в командной строке:

```
[packt@fedora ~]$ dnf group list --all | grep KDE
KDE Plasma Workspaces
KDE Plasma Mobile
... другие группы ...

[packt@fedora ~]$ sudo dnf groupinstall "KDE Plasma Workspaces"
Copr repo for PyCharm owned by phracek
1.5 kB/s | 2.4 kB      00:01   Fedora 41 - x86_64
5.5 MB/s | 34 MB      00:06   Fedora 41 openh264 (From Cisco) - x86_64
...
=====
Package                Architecture Version      Repository Size
=====
Installing group/module packages:
adwaita-gtk2-theme      x86_64      3.28-20.fc41 fedora      190 k
akonadi-server-mysql     x86_64      24.12.1-1.fc41 updates    12 k
... другие пакеты и зависимости ...
Installing Environment Groups:
KDE Plasma Workspaces
Installing Groups:
Administration Tools
base-graphical
Core
... другие группы ...

Transaction Summary
=====
Install 430 Packages
Total download size: 452 M
Installed size: 1.3 G
Is this ok [y/N]: y

Downloading Packages:
...
(1/430): accounts-qml-module-qt6-0.7^20231216... 154 kB/s | 87 kB 00:00
```

У DNF 5 немало и других отличий: например, у подкоманды `list` нет ключа `--all`, потому что соответствующий режим действует по умолчанию. Чтобы подробнее узнать о DNF 5, обратитесь к документации: dnf5.readthedocs.io. — *Примеч. ред.*

```

(2/430): SDL2_ttf-2.22.0-3.fc41.x86_64.rpm      74 kB/s |  44 kB 00:00
... другие пакеты...
-----
Total                                          7.7 MB/s | 452 MB 00:58
... дополнительные проверки ...

Running transaction
  Preparing      :                                1/1
  Installing    : kf6-kservice-6.10.0-1.fc41.x86_64 1/430
  Installing    : kf6-kglobalaccel-6.10.0-1.fc41.x86_64 2/430
  ... другие компоненты ...

Installed:
  NetworkManager-l2tp-1.20.20-1.fc41.x86_64
  NetworkManager-l2tp-gnome-1.20.20-1.fc41.x86_64
  ... другие компоненты ...
Complete!

```

Установка может занять некоторое время в зависимости от скорости интернет-соединения. Чтобы переключиться на KDE Plasma в качестве среды рабочего стола, нужно выйти из активного сеанса. После этого на экране входа в систему выберите своего пользователя, а затем в правом нижнем углу нажмите на значок шестеренки и выберите **Plasma**. Вы увидите два варианта: один для оконной системы, основанной на **Wayland**, а другой — для **X11**¹:

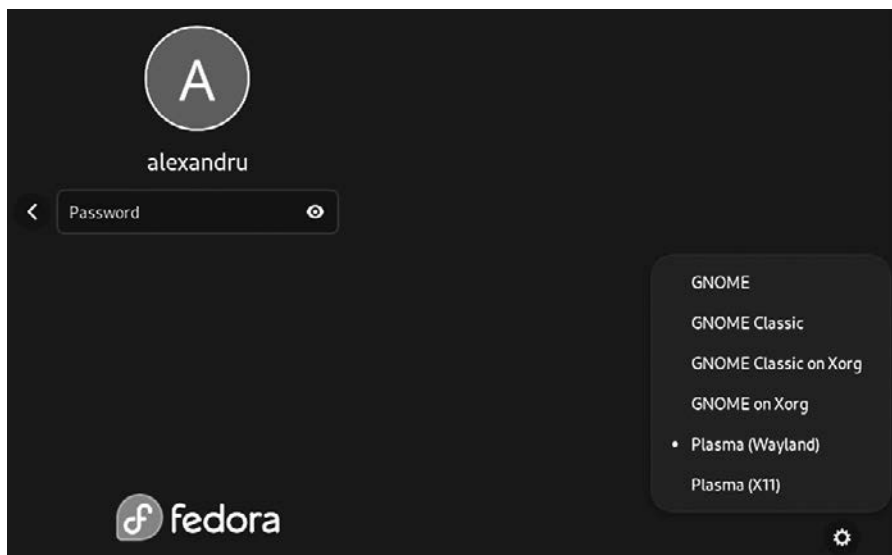


Рис. 3.3. Выбор Plasma(Wayland) на экране входа в систему

¹ Начиная с версии Fedora 40, KDE Plasma для Fedora поддерживается только с графическим сервером Wayland. — *Примеч. ред.*

Wayland — более новый графический сервер, который пока может поддерживаться в KDE не так хорошо, как в GNOME. Выбирайте то, что вам больше подходит.

Теперь можно войти в систему и работать с KDE Plasma на Fedora Workstation. На следующем снимке экрана показано приложение **Параметры системы (Info Center)**, которое работает под управлением KDE Plasma и показывает сведения об установленной версии ОС и оборудовании:

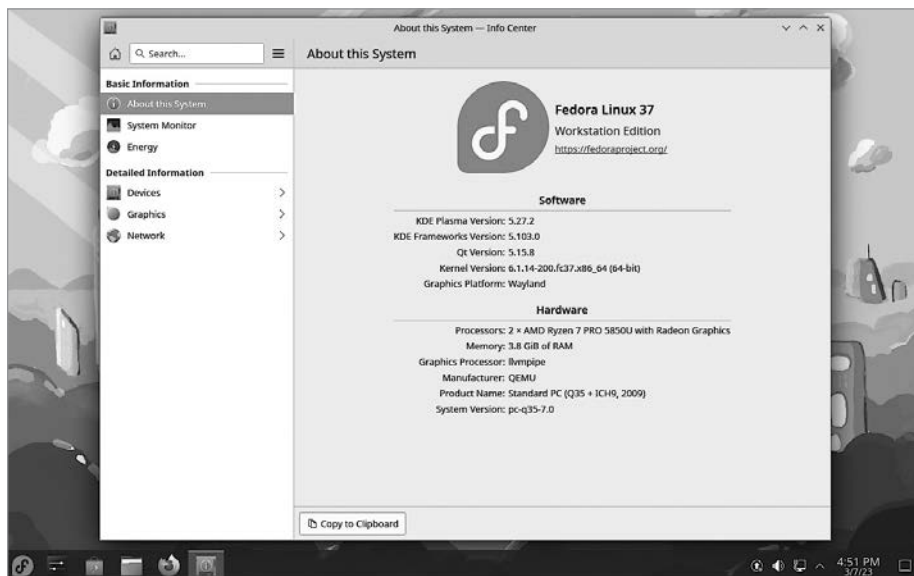


Рис. 3.4. Параметры системы в KDE Plasma на Fedora 37

Итак, вы узнали о том, как работать с пакетами в Linux и даже как установить новую среду рабочего стола. Этого достаточно, чтобы начать практиковаться с новой операционной системой. Теперь вы можете устанавливать новые приложения, конфигурировать их и настраивать свой дистрибутив так, как вам нужно.

Итоги

В этой главе вы научились работать с пакетами в Ubuntu, Fedora/AlmaLinux и openSUSE. Полученные навыки помогут вам управлять пакетами в любом дистрибутиве Linux. Вы узнали, как работать с пакетами deb и RPM, а также с такими новыми типами пакетов, как Flatpak и Snap. Эти знания пригодятся вам во всех дальнейших главах этой книги, а также в повседневной работе в качестве системного администратора и даже в свободное время, которое вы посвящаете своей операционной системе Linux.

В следующей главе мы поговорим о том, как управлять учетными записями и правами доступа пользователей. Вы познакомитесь с общими понятиями и изучите соответствующие инструменты на практике.

Вопросы

Теперь, когда у вас есть четкое представление о том, как управлять программными пакетами, предлагаем вашему вниманию несколько упражнений, которые помогут закрепить знания:

1. Составьте список всех пакетов, установленных в вашей системе.

Подсказка: попробуйте использовать команду `apt list --installed`.

2. Добавьте поддержку Flatpak в свою систему Ubuntu.

Подсказка: следуйте документации на flatpak.org.

3. Рассмотрите другие дистрибутивы и опробуйте их системы управления пакетами. Мы рекомендуем попрактиковаться с openSUSE, а если вы чувствуете себя уверенно, — то с Arch Linux.

Дополнительная литература

Чтобы получить дополнительную информацию о том, что рассматривалось в этой главе, обратитесь к официальной документации:

- Пакеты Snap: snapcraft.io/docs
- Пакеты Flatpak: docs.flatpak.org
- Дистрибутив openSUSE: doc.opensuse.org

4

Как управлять пользователями и группами

Linux — многопользовательская и многозадачная операционная система: это означает, что разные пользователи могут одновременно работать в системе и совместно обращаться к ресурсам платформы, а ядро при этом выполняет задачи каждого пользователя в конкурентном режиме независимо от остальных. Linux обеспечивает необходимые механизмы изоляции и безопасности, чтобы разные пользователи не могли получить доступ к файлам друг друга или удалить их.

Когда в системе несколько пользователей, в дело вступают права доступа. В этой главе мы поговорим о том, как устроены **права доступа** (они же **разрешения**) в Linux, и рассмотрим их основные категории — чтение, запись и выполнение. Вы также познакомитесь с понятием **суперпользователя (root)**, чья учетная запись имеет полный доступ к ресурсам операционной системы.

Мы будем подходить к изучаемым темам с практической стороны и осваивать ключевые понятия на конкретных упражнениях. В этой главе рассматриваются такие темы:

- Как управлять пользователями.
- Как управлять группами.
- Как управлять правами доступа.

Надеемся, что к концу главы вы будете уверенно работать с утилитами командной строки, которые позволяют создавать, изменять и удалять пользователей и группы, а также сможете грамотно управлять правами доступа к файлам и каталогам.

Давайте вкратце рассмотрим технические требования к этой главе.

Технические требования

Вам понадобится действующий дистрибутив Linux, который установлен либо в виртуальной машине, либо на настольной платформе. Если у вас его еще нет, то установить его поможет глава 1 «Как установить Linux». В последующей

главе мы будем использовать Ubuntu или Fedora, но большинство рассмотренных команд и примеров подходят и для любой другой платформы Linux.

Как управлять пользователями

В данном контексте **пользователь** — это любой человек или техническая система, которая использует компьютер или системные ресурсы. В простейшей форме *учетная запись* пользователя Linux идентифицируется именем и уникальным идентификатором, который известен как **UID**¹.

С чисто технической точки зрения в Linux бывают такие типы пользователей:

- **Обычные пользователи.** Учетные записи общего назначения предназначены в основном для живых пользователей, которые работают с приложениями и манипулируют файлами, имея ограниченный доступ к общесистемным ресурсам. Как правило, у обычных пользователей есть *оболочка для входа в систему и домашний каталог*.
- **Системные пользователи.** Эти учетные записи похожи на обычные за исключением того, что у них может не быть оболочки для входа в систему или домашнего каталога. Системные пользователи обычно соответствуют фоновым службам приложений — в основном по соображениям безопасности и для того, чтобы ограничить поверхность атаки, связанную с соответствующими ресурсами. Например, демон веб-сервера обрабатывает внешние запросы, и поэтому ему лучше работать под системной учетной записью, в идеале без входа в систему или полномочий суперпользователя. В этом случае возможные уязвимости на стороне сервера останутся строго изолированными в ограниченном пространстве соответствующего системного пользователя.
- **Суперпользователи.** Это привилегированные учетные записи, у которых есть полный доступ к системным ресурсам, в том числе право создавать, изменять и удалять пользователей. Примером суперпользователя является `root`.

В Linux только пользователь `root` или пользователи с привилегиями `sudo` могут создавать, изменять или удалять учетные записи пользователей.

Что такое sudo

В Linux `root` — это стандартная учетная запись суперпользователя, который может делать в системе все что угодно. В идеале по соображениям безопасности стоит как можно меньше работать от имени `root`. С помощью `sudo` Linux позволяет *повысить* привилегии обычного пользователя до уровня суперпользо-

¹ Сокр. от user identifier — «идентификатор пользователя» (*англ.*). В этой книге термины «пользователь» и «учетная запись» используются взаимозаменяемо. — *Примеч. пер.*

вателя, задействуя дополнительный уровень безопасности. Поэтому вместо того, чтобы работать под учетной записью `root`, пользователи Linux обычно применяют `sudo`.

`sudo` — это утилита командной строки, которая позволяет уполномоченным пользователям выполнять команды с привилегиями суперпользователя или другого пользователя (в зависимости от политики безопасности локальной системы). Название `sudo` первоначально расшифровывалось как *superuser do*, потому что программа была реализована так, чтобы выполнять действия исключительно с правами суперпользователя. Однако с тех пор функции утилиты расширились, и она позволяет замещать не только суперпользователя, но и другие (ограниченные) учетные записи, так что ее название также расшифровывают как *substitute user do*¹. Впрочем, чаще всего `sudo` по-прежнему рассматривают как замещение суперпользователя, потому что именно для этого эту команду в основном применяют в административных задачах Linux.

Большинство инструментов командной строки, с помощью которых можно управлять пользователями в Linux, требуют привилегий `sudo`, если только вы не работаете под учетной записью `root`. Однако получается, что если мы не хотим заходить в систему как `root`, то не сможем выполнять дальнейшие упражнения из этой главы — и, в частности, создавать пользователей, — пока у нас не будет учетной записи с привилегиями суперпользователя. Поэтому давайте сначала разберемся с этой проблемой «курицы и яйца».

В большинстве дистрибутивов Linux при установке создается еще одна учетная запись с привилегиями суперпользователя, помимо `root`. Как уже отмечалось, это делается для того, чтобы обеспечить дополнительный уровень безопасности для операций с повышенными привилегиями. Самый простой способ проверить, есть ли у вашей учетной записи доступ к `sudo`, — запустить в терминале такую команду:

```
sudo -v
```

Согласно справочной странице `sudo` (`man sudo`), ключ `-v` заставляет утилиту обновить кэшированные учетные данные пользователя и запросить аутентификацию, если их срок действия истек.

Если у учетной записи (например, `julian`) нет привилегий суперпользователя на локальном компьютере (например, `neptune`), то предыдущая команда выдаст такое (или похожее) сообщение об ошибке:

```
Sorry, user julian may not run sudo on neptune.2
```

¹ Подменить пользователя (и) выполнить. — *Примеч. пер.*

² Сожалеем, но пользователь `julian` не уполномочен запускать `sudo` на компьютере `neptune`. — *Примеч. пер.*

В современных дистрибутивах Linux команда `sudo` обычно предоставляет повышенные права на ограниченное время. Например, в Ubuntu этот период составляет 15 минут, после чего снова нужно аутентифицироваться. Если вы повторно вызываете `sudo`, система может не запрашивать пароль, пока не истечет срок кэширования учетных данных.

Если у вас нет стандартной учетной записи суперпользователя, всегда можно использовать контекст `root`, чтобы создать новых пользователей и предоставить им доступ к `sudo`. Мы подробнее поговорим об этом в разделе «Как настраивать учетные записи суперпользователей» далее в этой главе.

Теперь давайте посмотрим, как создавать, изменять и удалять пользователей.

Как создавать, изменять и удалять учетные записи пользователей

В этом разделе мы рассмотрим несколько инструментов командной строки и некоторые типичные задачи управления пользователями. Мы разберем соответствующие команды и процедуры на примере Ubuntu и Fedora, но все те же принципы применимы и к любому другому дистрибутиву Linux. На некоторых платформах отдельные инструменты командной строки могут отличаться или быть недоступными: например, в Alpine Linux нет команды `useradd`, и вместо нее следует использовать `adduser`. Пожалуйста, обратитесь к документации своего дистрибутива Linux, чтобы найти эквивалентные команды.

Как создавать учетные записи пользователей

Чтобы создавать пользователей, можно использовать команду `useradd` или `adduser`. В некоторых дистрибутивах Linux (например, Debian или Ubuntu) рекомендуется применять именно `adduser`, а не низкоуровневую утилиту `useradd`. В этом разделе мы рассмотрим оба варианта.

`adduser` — это сценарий на Perl, который обращается к `useradd`; по сути, это обертка команды `useradd` с удобным мастером конфигурации. Оба инструмента установлены по умолчанию в Ubuntu и Fedora. Давайте вкратце рассмотрим каждый из них.

Как создавать учетные записи пользователей с помощью `useradd`

Синтаксис команды `useradd` устроен так:

```
useradd [ключи] имя_пользователя
```

В самом простом варианте следующая команда создает учетную запись пользователя `julian`:

```
sudo useradd julian
```


Сведения об учетных записях хранятся в файле `/etc/passwd`. Вот соответствующие данные пользователя `julian`:

```
packt@neptune:~$ sudo cat /etc/passwd | grep julian
julian:x:1001:1001:./home/julian:/bin/sh
```

Давайте разберемся, как устроены учетные записи пользователей. Каждая запись состоит из полей, которые разделены двоеточиями (:). Эти поля таковы:

<code>julian</code>	Имя пользователя
<code>x</code>	Пароль пользователя. Буква <code>x</code> означает, что зашифрованный пароль хранится в <code>/etc/shadow</code>
<code>1001</code>	Идентификатор пользователя (UID)
<code>1001</code>	Идентификатор группы пользователя (GID)
<code>::</code>	Поле GECOS — в нашем случае пустое, но может содержать отображаемое имя и другие данные — см. далее в этом разделе
<code>/home/julian</code>	Домашний каталог пользователя
<code>/bin/sh</code>	Командная оболочка пользователя для входа в систему по умолчанию

Поле GECOS¹ содержит атрибуты, которые разделены запятыми и отражают общую информацию об учетной записи пользователя — например, настоящее имя, название компании и номер телефона. В Linux это пятое поле в учетной записи. Дополнительную информацию можно найти в статье en.wikipedia.org/wiki/Gecos_field.

Чтобы извлечь ту же запись о пользователе, можно также запустить команду `getent`:

```
getent passwd julian
```

Чтобы просмотреть идентификаторы пользователя (UID) и соответствующей группы (GID), а также идентификаторы групп, в которых состоит пользователь, используйте команду `id`:

```
packt@neptune:~$ id julian
uid=1001(julian) gid=1001(julian) groups=1001(julian)
```

¹ Название поля GECOS — аббревиатура от General Comprehensive Operating System (комплексная операционная система общего назначения), ранее General Electric Comprehensive Operating Supervisor (комплексный операционный супервизор компании General Electric). Так называлось семейство операционных систем для мэйнфреймов, которые эксплуатировались с 1960-х годов. Некоторые особенности этих систем (например, поле GECOS) сохранились в последующих ОС семейства Unix и Linux. — *Примеч. ред.*

Если вызвать `useradd` без ключей, эта команда создает учетную запись (например, `julian`) и заполняет некоторые ее поля значениями по умолчанию (как перечислено выше), в то время как другие поля остаются пустыми: например, в нашем примере у пользователя нет полного имени или пароля. Кроме того, хотя для учетной записи указан домашний каталог по умолчанию (например, `/home/julian`), фактически этот каталог будет создан в файловой системе только после того, как вы запустите команду `useradd` с ключом `-m` (`--create-home`), например:

```
sudo useradd -m julian
```

Без домашнего каталога обычные пользователи не смогут сохранять свои файлы в своем собственном месте системы. С другой стороны, некоторые системные учетные записи могут обойтись без домашнего каталога, потому что у них нет оболочки для входа в систему. Допустим, сервер баз данных (такой как PostgreSQL) может работать от имени системной учетной записи, отличной от `root` (например, `postgres`), которой нужен доступ только к ресурсам баз данных в определенных каталогах (таких как `/var/lib/pgsql`), и этот доступ контролируется с помощью других механизмов разрешений (например, SELinux).

Если бы нам понадобилось указать полное (отображаемое) имя нашего обычного пользователя, то команду следовало бы изменить так:

```
sudo useradd -m -c "Julian" julian
```

Ключ `-c` (или `--comment`) команды `useradd` позволяет задать *комментарий* (он же поле GECOS) с несколькими значениями, разделенными запятыми. В нашем случае мы указали полное имя (`Julian`). Если вас интересуют подробности, запустите `man useradd` или `useradd --help`.

У нового пользователя еще нет пароля, а значит, он не сможет войти в систему (например, через графический интерфейс или по SSH). Чтобы создать пароль для учетной записи `julian`, можно вызвать команду `passwd`:

```
packt@neptune:~$ sudo passwd julian
New password:
Retype new password:
passwd: password updated successfully1
```

Команда `passwd` запрашивает пароль нового пользователя. После того как вы создали пароль, соответствующая запись добавляется в файл `/etc/shadow`. В нем хранятся защищенные хеши паролей (а не сами пароли!) для каждого пользователя. Доступ к содержимому этого файла есть только у суперпользователей. Вот как можно двумя способами просмотреть данные для пользователя `julian`²:

¹ Пароль успешно обновлен. — *Примеч. пер.*

² Помимо хешированного пароля, каждая строка в файле `/etc/shadow` содержит другие сведения об учетной записи, в частности ее имя, дату последней смены пароля и срок

```
packt@neptune:~$ sudo getent shadow julian
julian:$y$j9T$SKGCTh6m06LJTk9JTeIZ2.$RRNq/tXCmnyhAXvx8WyVL83DRgADv3Zq4qe
RhE72J82:20126:0:99999:7:::

packt@neptune:~$ sudo cat /etc/shadow | grep julian
julian:$y$j9T$SKGCTh6m06LJTk9JTeIZ2.$RRNq/tXCmnyhAXvx8WyVL83DRgADv3Zq4qe
RhE72J82:20126:0:99999:7:::
```

Обычно после того, как пароль установлен, пользователь может войти в систему (например, через SSH или графический интерфейс). Если в дистрибутиве Linux есть графический интерфейс, то имя нового пользователя появится на экране входа.

Как уже отмечалось, команда `useradd` предоставляет средства для тонкого низкоуровневого управления учетными записями, но иногда уместнее более «человекоориентированный» подход. Давайте познакомимся с командой `adduser`.

Как создавать учетные записи пользователей с помощью `adduser`

Команда `adduser` — это обертка для `useradd`, написанная на Perl. Синтаксис этой команды таков:

```
adduser [ключи] имя_пользователя
```

Чтобы запускать `adduser`, вам может понадобиться `sudo` и пароль суперпользователя. `adduser` запросит пароль нового пользователя и другую информацию о нем, как показано в следующем примере.

Давайте создадим новую учетную запись пользователя `alex` с помощью `adduser`:

```
packt@neptune:~$ sudo adduser alex
info: Adding user `alex' ...
      # Добавляем пользователя alex
info: Selecting UID/GID from range 1000 to 59999 ...
      # Выбираем UID/GID из диапазона от 1000 до 59999
info: Adding new group `alex' (1002) ...
      # Добавляем новую группу alex (GID 1002)
info: Adding new user `alex' (1002) with group `alex (1002)' ...
      # Добавляем нового пользователя alex (UID 2002) с группой alex (GID 2002)
info: Creating home directory `/home/alex' ...
      # Создаем домашний каталог /home/alex
info: Copying files from `/etc/skel' ...
      # Копируем файлы из /etc/skel
New password:          # Новый пароль
Retype new password:   # Повторите новый пароль
passwd: password updated successfully
      # Сообщение команды passwd: пароль успешно обновлен
Changing the user information for alex
```

действия учетной записи. — *Примеч. ред.*

¹ Перевод в виде фиктивных комментариев Bash приводится для удобства читателей и отсутствует в реальном выводе команды `adduser`. — *Примеч. пер.*

```

# Изменяем сведения о пользователе alex
Enter the new value, or press ENTER for the default
# Введите новое значение или нажмите Enter,
# чтобы применить значение по умолчанию
Full Name []: # Полное имя
Room Number []: # Номер кабинета
Work Phone []: # Рабочий телефон
Home Phone []: # Домашний телефон
Other []: # Прочее
Is the information correct? [Y/n] Y
# Эти сведения верны? [Да/нет]
info: Adding new user `alex' to supplemental / extra groups `users' ...
info: Adding user `alex' to group `users' ...
# Добавляем пользователя alex в группу users

```

В Fedora предыдущая команда `adduser` просто выполнится, не запрашивая у пользователя пароль или другие данные.

С помощью `getent` можно увидеть соответствующую учетную запись в файле `/etc/passwd`:

```

packt@neptune:~$ getent passwd alex
alex:x:1002:1002::,:/home/alex:/bin/bash

```

В предыдущих примерах мы создавали учетные записи обычных пользователей. Администраторы или суперпользователи также могут повысить привилегии обычного пользователя до суперпользователя. В следующем разделе мы расскажем, как это сделать.

Как настраивать учетные записи суперпользователей

Когда обычный пользователь получает право запускать `sudo`, он становится суперпользователем. Для дальнейших упражнений предположим, что у нас есть обычная учетная запись, которую мы создали с помощью любого метода из раздела «Как создавать учетные записи пользователей».

Чтобы повысить уровень пользователя до суперпользователя, его нужно включить в группу `sudo`¹. В Linux `sudo` — это зарезервированная системная группа для пользователей с повышенными полномочиями (в том числе `root`). Чтобы сделать учетную запись `julian` суперпользователем, ее нужно просто добавить в группу `sudo`, как показано ниже (в Ubuntu):

```
sudo usermod -aG sudo julian
```

Здесь ключи `-aG` предписывают команде добавить (`-a`, `--append`) пользователя в указанную группу (`-G`, `--group`) — в нашем случае группу `sudo`.

¹ В англоязычной литературе суперпользователи (`superusers`) также называются `sudoers` (букв. «судошники»). — *Примеч. пер.*

Чтобы проверить, что учетная запись теперь является суперпользователем, сначала с помощью команды `id` убедитесь, что сведения о пользователе отражают членство в `sudo`:

```
packt@neptune:~$ id julian
uid=1001(julian) gid=1001(julian) groups=1001(julian),27(sudo)
```

Этот вывод показывает, что пользователь входит в группу `sudo`, идентификатор (GID) которой равен 27.

Чтобы проверить, что у пользователя `julian` есть доступ к `sudo`, запустите такую команду:

```
su - julian
```

Обычно эта команда запрашивает пароль пользователя `julian`, и если удастся войти в систему, это свидетельствует о том, что учетная запись принадлежит суперпользователю. Пользователь `julian` также может запустить команду `sudo -v` в своем терминале, чтобы убедиться, что у него есть доступ к `sudo`. Дополнительные сведения о привилегиях суперпользователя см. в разделе «Что такое `sudo`» в начале этой главы.

Если в системе много пользователей, системному администратору может понадобиться просмотреть их список или сведения о них. В следующем разделе мы покажем, как это сделать.

Как просматривать сведения о пользователях

У суперпользователя есть несколько способов просмотреть все учетные записи, которые существуют в системе. Мы уже говорили о том, что сведения о пользователях хранятся в файлах `/etc/passwd` и `/etc/shadow`. Можно не только напрямую просматривать эти файлы, но и разобрать их на отдельные поля. Например, такая команда извлекает только имена пользователей из `/etc/passwd`:

```
cat /etc/passwd | cut -d: -f1 | less
```

А так можно извлечь те же имена из файла `/etc/shadow`:

```
sudo cat /etc/shadow | cut -d: -f1 | less
```

В каждой из предыдущих команд мы считали содержимое указанных файлов с помощью `cat`, а затем передали результат в утилиту выборки полей `cut` (указав разделитель `:`) и выбрали первое поле (`-f1`). Наконец, мы запросили постраничный вывод результатов с помощью команды `less` (чтобы выйти из нее, нажмите `q`).

Обратите внимание, что для файла `/etc/shadow` пришлось использовать `sudo`: он доступен только суперпользователям, потому что хеш-суммы паролей относятся к конфиденциальной информации. Чтобы получить сведения об учетных записях, можно также использовать команду `getent`.

Следующая команда перечисляет все учетные записи, которые есть в системе, на основе данных из файла `/etc/passwd`:

```
getent passwd
```

Ту же информацию можно получить из `/etc/shadow`:

```
sudo getent shadow
```

И в том и в другом случае вывод `getent` можно перенаправить в `cut -d: -f1`, чтобы перечислить только имена пользователей, например:

```
packt@neptune:~$ sudo getent shadow | cut -d: -f1 | less | column
root                www-data            sshd
daemon             backup              syslog
bin                 list                uuidd
sys                 irc                 tcpdump
sync                gnats               tss
games               nobody              landscape
man                 _apt                usbmux
lp                  systemd-network    packt
mail                systemd-resolve     lxd
news                messagebus          fwupd-refresh
uucp                systemd-timesync    julian
proxy               pollinate           alex
```

Иногда администраторам или суперпользователям требуется изменить те или иные свойства учетной записи — например, пароль или срок его действия, полное имя или оболочку для входа. Далее мы рассмотрим самые распространенные способы сделать это.

Как редактировать учетные записи пользователей

Чтобы изменить свойства пользователя, суперпользователь может запустить команду `usermod` со следующим синтаксисом:

```
usermod [ключи] имя_пользователя
```

В этом разделе мы продолжим работать с учетной записью `julian`, которую создали ранее с помощью простейшего вызова команды `useradd`. Как отмечалось в предыдущем разделе, в соответствующей строке в `/etc/passwd` пока нет полного имени пользователя, и пароля у него тоже нет.

Давайте изменим такие свойства пользователя `julian`:

	Было (по умолчанию)	Стало
Полное имя	пустое	Julian
Домашняя папка	/home/julian	/local/julian
Оболочка для входа в систему	/bin/sh	/bin/bash

Чтобы отредактировать все эти данные, можно ввести такую команду:

```
sudo usermod -c "Julian" -d /local/julian -m -s /bin/bash julian
```

Вот за что отвечает каждый ключ этой команды:

Ключ	Длинная форма	Назначение
-	-	Полное имя пользователя
-d	-home	Новый домашний каталог пользователя
-m	-move	Переместить содержимое текущего домашнего каталога в новое место
-s	-shell	Оболочка для входа пользователя в систему

С помощью команды `getent` можно посмотреть, что изменилось:

```
packt@neptune:~$ getent passwd julian
julian:x:1001:1001:Julian:/local/julian:/bin/bash
```

Вот еще несколько примеров того, какие свойства пользователя можно изменить с помощью `usermod`.

Как изменить имя пользователя

Ключ `-l (--login)` команды `usermod` задает новое имя пользователя. Следующая команда изменяет имя пользователя с `julian` на `balog` (то есть с имени на фамилию):

```
sudo usermod -l "balog" julian
```

В среде реальной эксплуатации нам, возможно, пришлось бы дополнить предыдущую команду и изменить также отображаемое имя и домашний каталог пользователя, чтобы они согласовывались с именем учетной записи. В предыдущем примере в разделе «Как редактировать учетные записи пользователей» мы показывали ключи `-d` и `-m`, которые позволяют внести такие изменения.

Как блокировать или разблокировать учетные записи

Суперпользователь или администратор может на время или навсегда заблокировать определенного пользователя с помощью ключа `-L (--lock)`, например:

```
sudo usermod -L julian
```

Если запустить предыдущую команду, то после этого пользователь `julian` не сможет войти в систему. Если пользователь попытается подключиться к компьютеру под управлением Linux по SSH, он получит сообщение об ошибке вида

«**Permission denied, please try again**»¹. Кроме того, если у системы Linux есть графический интерфейс, то соответствующее имя пользователя пропадет с экрана входа в систему.

Чтобы разблокировать пользователя, используйте ключ `-U` (`--unlock`), например:

```
sudo usermod -U julian
```

После этого пользователь снова получит доступ к системе.

Чтобы подробнее узнать об утилите `usermod`, ознакомьтесь с ее справочной страницей (`man usermod`) или встроенной справкой (`usermod --help`).

Хотя специалисты рекомендуют изменять свойства учетных записей именно с помощью утилиты командной строки `usermod`, кому-то может показаться проще вручную отредактировать файл `/etc/passwd`. В следующем разделе показано, как это сделать.

Как редактировать учетные записи пользователей в `/etc/passwd`

Суперпользователи также могут вручную редактировать файл `/etc/passwd`, чтобы изменить свойства учетных записей. Хотя это можно делать с помощью вашего любимого текстового редактора (например, `nano`), безопаснее будет использовать утилиту командной строки `vipw`. Она устанавливает необходимые блокировки, чтобы предотвратить возможное повреждение данных — например, если суперпользователь редактирует записи в то же время, когда кто-то из обычных пользователей меняет свой пароль.

Команда `vipw` предлагает выбрать предпочтительный текстовый редактор (допустим, `nano` или `vim`), а затем открывает в нем файл `/etc/passwd`:

```
sudo vipw
```

Чтобы изменить свойства учетной записи `julian`, можно отредактировать следующую строку:

```
julian:x:1001:1001:Julian,,,:/home/julian:/bin/bash
```

Структура полей, разделенных двоеточием (`:`), рассматривалась ранее в разделе «Как создавать учетные записи пользователей с помощью `useradd`». Если вы открыли файл `/etc/passwd` в текстовом редакторе, то можете отредактировать любое из этих полей вручную, и это приведет к тем же изменениям, что и при использовании соответствующей команды `usermod`.

Чтобы получить дополнительную информацию об утилите `vipw`, можно обратиться к соответствующей справочной странице (`man vipw`).

¹ Отказано в доступе; пожалуйста, повторите попытку. — *Примеч. пер.*

Другие довольно распространенные задачи по управлению пользователями связаны с паролями: например, иногда нужно сменить пароль или задать срок его действия. Хотя `usermod` позволяет изменить любой пароль с помощью ключа `-p` (`--password`), для этого требуется зашифрованная хеш-сумма пароля, а не сам пароль в открытом виде. Вряд ли имеет смысл утруждаться, чтобы сгенерировать эту хеш-сумму, потому что есть более простой способ — сменить пароль с помощью утилиты `passwd`.

Суперпользователь (администратор) может изменить пароль учетной записи (например, пользователя `julian`) с помощью такой команды:

```
sudo passwd julian
```

После этого система предложит ввести и подтвердить новый пароль для указанного пользователя. А чтобы изменить срок действия пароля, используется команда `chage`¹. Например, чтобы пароль пользователя `julian` действовал в течение 30 дней, запустите такую команду:

```
sudo chage -M 30 julian
```

В результате пользователю `julian` понадобится менять свой пароль каждый месяц. Периодичность смены пароля зависит от политики управления паролями, которая настраивается на уровне всей системы и находится в файле `/etc/login.defs`, в разделе **Password aging controls**². Если изменить этот файл, обновленная политика паролей распространится на всех пользователей.

Еще одна распространенная практика — потребовать от пользователя сменить пароль при первом входе в систему. Например, для учетной записи `julian` это можно сделать с помощью такой команды:

```
sudo chage -d 0 julian
```

Иногда администратору требуется удалить из системы те или иные учетные записи. В следующем разделе рассказывается, как это сделать.

Как удалять учетные записи пользователей

Самый распространенный способ удалить учетные записи пользователей из системы — применить инструмент командной строки `userdel`. Общий синтаксис этой команды таков:

```
userdel [ключи] имя_пользователя
```

¹ Это не опечатка: команда называется `chage`, а не `change`. Ее название — сокр. от `change age` (изменить устаревание). — *Примеч. пер.*

² Управление устареванием паролей. — *Примеч. пер.*

Например, чтобы удалить пользователя `julian`, суперпользователю нужно запустить такую команду:

```
sudo userdel -f -r julian
```

Здесь используются такие ключи:

Ключ	Длинная форма	Назначение
<code>-f</code>	<code>-force</code>	Удалить все файлы в домашнем каталоге пользователя, даже если он ими не владеет
<code>-r</code>	<code>-remove</code>	Удалить домашний каталог пользователя и его почтовый ящик

Команда `userdel` удаляет из системы данные, которые связаны с пользователем, включая его домашний каталог (если использовать ключ `-f` или `--force`) и соответствующие записи в файлах `/etc/passwd` и `/etc/shadow`.

Учетные записи можно удалять и по-другому. В следующем разделе мы представим альтернативный способ, который может пригодиться в некоторых специфических сценариях очистки.

Как удалять учетные записи пользователей через файлы `/etc/passwd` и `/etc/shadow`

Чтобы удалить учетную запись, суперпользователь может вручную отредактировать файлы `/etc/passwd` и `/etc/shadow` и удалить из них обеих строки, которые относятся к целевому пользователю (например, `julian`). Имейте в виду, что нужно согласованно изменить оба файла, чтобы учетная запись полностью удалась и система не осталась в противоречивом состоянии.

В качестве упражнения откройте в текстовом редакторе файл `/etc/passwd` с помощью `vi`:

```
sudo vi /etc/passwd
```

Удалите строку, которая начинается с имени пользователя `julian`:

```
julian:x:1001:1001:Julian,,,:/home/julian:/bin/bash
```

Затем откройте файл `/etc/shadow`, вызвав команду `vi` с ключом `-s` (`--shadow`):

```
sudo vi -s /etc/shadow
```

Также удалите строку, которая начинается с имени пользователя `julian`:

```
julian:$6$xDdd7Eay/RKYjeTm$Sf.../:18519:0:99999:7:::
```

После того как вы отредактировали оба файла, может потребоваться еще удалить домашний каталог того же пользователя:

```
sudo rm -rf /home/julian
```

Чтобы получить дополнительную информацию об утилите `userdel`, ознакомьтесь с соответствующей справочной страницей (`man userdel`) или встроенной справкой (`userdel --help`).

Понятия и команды, которые мы изучали до сих пор, касались исключительно того, как управлять отдельными пользователями системы. Когда нескольким пользователям нужно предоставить одинаковые полномочия или права доступа, их можно объединить в группу. Группы — это организационные единицы, которые можно создавать, изменять или удалять, а кроме того, можно регулировать членство пользователей в группах. Следующий раздел посвящен тому, как управлять группами.

Как управлять группами

Группы в Linux позволяют агрегировать пользователей. Проще говоря, группа — это совокупность пользователей, объединенных общим атрибутом. Например, в одну группу могут входить линейные сотрудники, в другую — разработчики, в третью — менеджеры и т. д. В Linux каждая группа характеризуется уникальным идентификатором группы (GID). Всем пользователям в одной группе соответствует один и тот же GID.

С точки зрения пользователя, группы бывают двух типов:

- **Основная (первичная) группа.** Исходная группа пользователя (по умолчанию) для входа в систему.
- **Дополнительные (вторичные) группы.** Перечень других групп, в которых состоит пользователь.

Каждый пользователь Linux обязательно состоит в какой-то одной основной группе и при этом может входить в одну или несколько дополнительных групп или не входить в них вовсе.

С практической точки зрения можно рассматривать группы как контекст, который предоставляет избранным пользователям определенные права доступа для совместной работы. Допустим, у группы `developers` (разработчики) есть доступ к ресурсам, которые предназначены для разработчиков, и каждый пользователь в этой группе может обращаться к этим ресурсам. А у пользователей, которые не входят в группу *разработчиков*, может не быть такого доступа до тех пор, пока они не пройдут аутентификацию с помощью группового пароля (если он задан).

В следующем разделе мы рассмотрим подробные примеры того, как управлять группами и регулировать членство пользователей в них. Большинство соответствующих команд требуют привилегий суперпользователя или `sudo`.

Как создавать, изменять и удалять группы

Хотя здесь мы уделяем основное внимание операциям над группами, для некоторых задач все же понадобятся команды, которые относятся к отдельным

пользователям. Создавать, изменять и удалять группы позволяют утилиты командной строки `groupadd`, `groupmod` и `groupdel` соответственно. А уже знакомые вам команды `useradd` и `usermod` можно запускать с особыми ключами, чтобы ассоциировать пользователей с группами. Кроме того, вы познакомитесь с `grasswd` — инструментом командной строки, который предназначен для административных операций над группами и объединяет задачи, связанные с пользователями и группами.

Давайте рассмотрим, как создавать, изменять и удалять группы, а также как управлять членством пользователей в них.

Как создать группу

Чтобы создать новую группу, суперпользователю нужно вызвать утилиту командной строки `groupadd`. Вот ее основной синтаксис:

```
groupadd [ключи] имя_группы
```

Давайте создадим новую группу `developers` с настройками по умолчанию:

```
sudo groupadd developers
```

Сведения о группе хранятся в файле `/etc/group`. Вот соответствующие данные для группы `developers`:

```
packt@neptune:~$ cat /etc/group | grep developers
developers:x:1003:
```

Запись для каждой группы состоит из нескольких полей, разделенных двоеточием (:). В нашем случае поля таковы:

<code>developers</code>	Название группы
<code>x</code>	Маркер зашифрованного пароля (хеш-сумма пароля хранится в файле <code>/etc/gshadow</code>)
<code>1003</code>	Идентификатор группы (GID)

Чтобы получить информацию о группе, можно также использовать команду `getent`:

```
getent group developers
```

Суперпользователь может создать группу с определенным GID, вызвав команду `groupadd` с ключом `-g (--gid)`. Например, такая команда создает группу `developers` (если она не существовала ранее) с GID `1200`:

```
sudo groupadd -g 1200 developers
```

Чтобы узнать об утилите `groupadd` подробнее, обратитесь к соответствующей справочной странице (`man groupadd`).

Сведения о группах, подобно сведениям о пользователях, хранятся в двух файлах: файл `/etc/group` содержит общую информацию о каждой группе, а в файле `/etc/gshadow` хранятся зашифрованные хеш-суммы паролей для групп. Давайте вкратце рассмотрим групповые пароли.

Как устроены групповые пароли

Если создать группу с помощью `groupadd` без ключей (например, `groupadd developers`), то она возникнет без пароля. Хотя `groupadd` позволяет указать зашифрованный пароль (с помощью ключа `-p`, или `--password`), в этом случае понадобится приложить дополнительные усилия, чтобы сгенерировать защищенную хеш-сумму пароля. Гораздо проще и безопаснее задать групповой пароль с помощью утилиты командной строки `gpasswd`.

`gpasswd` — это инструмент командной строки, который помогает выполнять повседневные задачи администрирования групп.

Следующая команда задает пароль для группы `developers`. Она предлагает ввести и подтвердить пароль:

```
packt@neptune:~$ sudo gpasswd developers
Changing the password for group developers
New Password:
Re-enter new password:
```

Групповой пароль служит для того, чтобы защитить доступ к ресурсам группы. Такой пароль небезопасен по своей природе, если он известен членам группы, однако администратор Linux может сохранить его в тайне, что не мешает членам группы совместно работать в общем контексте безопасности.

Вкратце объясним, как функционирует эта схема. Когда член группы (например, группы `developers`) входит в эту группу (с помощью команды `newgrp`), он допускается туда без пароля. Но когда в группу пытаются войти пользователи, которые в ней не состоят, система запрашивает групповой пароль.

В общем случае в группе могут быть администраторы и другие члены. Участники группы, которые являются ее администраторами и вошли в группу, могут вызывать `gpasswd` без запроса пароля. Кроме того, администраторам групп не нужны привилегии суперпользователей, чтобы выполнять административные задачи в своих группах.

Мы рассмотрим `gpasswd` подробнее в следующих разделах, где поговорим о том, как управлять группами, а также как добавлять и удалять пользователей из

групп. А пока давайте сосредоточим внимание на уровне групп и посмотрим, как редактировать их свойства.

Как редактировать свойства группы

Самый распространенный способ редактировать свойства группы — это утилита командной строки `groupmod`. Вот ее синтаксис:

```
groupmod [ключи] имя_группы
```

Чаще всего требуется редактировать `GID`, имя и пароль группы. Давайте рассмотрим каждую из этих операций. Предположим, что ранее мы создали группу, которая называется `developers`, а ее `GID` равен `1003`.

Чтобы изменить `GID` на `1200`, суперпользователь может вызвать команду `groupmod` с параметром `-g` (`--gid`):

```
sudo groupmod -g 1200 developers
```

Чтобы изменить название группы с `developers` на `devops`, можно использовать ключ `-n` (`--new-name`):

```
sudo groupmod -n devops developers
```

Если вы хотите убедиться, что произошли нужные изменения, вызовите команду `getent` для группы с новым именем `devops`:

```
packt@neptune:~$ getent group devops
devops:x:1200:
```

Чтобы изменить групповой пароль для `devops`, проще всего использовать `gpsswd`. Эта команда предложит ввести и подтвердить новый пароль:

```
packt@neptune:~$ sudo gpsswd devops
Changing the password for group devops1
New Password:
Re-enter new password:
```

Чтобы удалить групповой пароль, ту же команду `gpsswd` нужно вызвать с ключом `-r`, (`--remove-password`), например:

```
sudo gpsswd -r devops
```

Поскольку эта команда не выводит никаких сообщений, после этого вы просто вернетесь в оболочку.

Чтобы получить дополнительную информацию об утилитах `groupmod` и `gpsswd`, обратитесь к их справочным страницам (`man groupmod` и `man gpsswd`) или просто вызовите каждую команду с ключом `-h` (`--help`).

¹ Изменяется пароль для группы `devops`. — *Примеч. пер.*

Далее мы рассмотрим, как удалять группы.

Как удалить группу

Удалять группы позволяет утилита командной строки `groupdel`. Вот ее синтаксис:

```
groupdel [ключи] имя_группы
```

По умолчанию Linux обеспечивает ссылочную целостность между пользователями и их основными (первичными) группами. Нельзя удалить группу, которая является основной для некоторых пользователей, если прежде не отвязать их от нее. Другими словами, по умолчанию Linux не допускает, чтобы пользователи оставались с «неприкаянными» первичными GID.

Например, когда мы создали пользователя `julian`, он был автоматически включен в основную группу `julian`. Затем мы добавили его в группу `sudoers`.

Давайте попробуем добавить пользователя `julian` в группу `devops`. Суперпользователь может запустить команду `usermod` с ключом `-g (--gid)`, чтобы *сменить* основную группу пользователя. Эту команду нужно вызывать по отдельности для каждой учетной записи.

Давайте попробуем удалить пользователя `julian` из его основной группы `julian`. Сначала уточним текущие данные пользователя:

```
packt@neptune:~$ id julian
uid=1003(julian) gid=1003(julian) groups=1003(julian),27(sudo)
```

Теперь давайте сменим основную группу пользователя `julian` на `devops`. Для этого достаточно указать либо GID, либо имя группы после ключа `-g (--gid)`. Если бы такой группы ранее не было в системе, команда завершилась бы неудачей. Запустим такие команды:

```
packt@neptune:~$ sudo usermod -g devops julian
packt@neptune:~$ id julian
uid=1003(julian) gid=1200(devops) groups=1200(devops),27(sudo)
```

Как видите, теперь пользователь `julian` состоит в основной группе `devops`.

Наконец, давайте попробуем удалить группу `devops`. Это приводит к ошибке, потому что нельзя удалить непустую группу:

```
packt@neptune:~$ sudo groupdel devops
groupdel: cannot remove the primary group of user 'julian'1
```

Суперпользователь может *принудительно* удалить первичную группу, если вызовет `groupdel` с ключом `-f (--force)`, но делать этого не рекомендуется. Дело в том, что в результате появятся пользователи с «подвешенными» первичными

¹ Сообщение команды `groupdel`: не удастся удалить первичную группу пользователя `julian`. — *Примеч. пер.*

GID, а это потенциальная брешь в системе безопасности. Обслуживать и удалять таких пользователей тоже будет проблематично.

Чтобы удалить группу `devops` без неприятных последствий, для пользователя `julian` нужно назначить другую основную группу. Можно снова связать его с группой `julian`, а затем удалить группу `devops`, потому что теперь она пуста:

```
packt@neptune:~$ sudo usermod -g julian julian

packt@neptune:~$ id julian
uid=1003(julian) gid=1003(julian) groups=1003(julian),27(sudo)

packt@neptune:~$ sudo groupdel devops
packt@neptune:~$ getent group devops1
```

Чтобы получить дополнительную информацию об утилите `groupdel`, обратитесь к соответствующей справочной странице (`man groupdel`) или просто вызовите `groupdel --help`.

Как редактировать свойства группы через `/etc/group`

Администратор также может вручную править файл `/etc/group`, чтобы изменять данные о группах, редактируя соответствующие строки. Хотя это можно делать с помощью любого текстового редактора (например, `nano`), мы рекомендуем использовать более безопасную утилиту командной строки `vigr`. Она похожа на `vipw` (с помощью которой мы редактировали `/etc/passwd`) и устанавливает защитные блокировки, чтобы данные не повредились, если разные процессы изменяют их одновременно.

Следующая команда открывает файл `/etc/group` для редактирования, а также запрашивает предпочтительный текстовый редактор (например, `nano` или `vim`):

```
sudo vigr
```

Например, чтобы изменить настройки группы `developers`, можно отредактировать следующую строку:

```
developers:x:1200:julian,alex
```

Если удалить группу с помощью команды `vigr`, она также предложит удалить соответствующую запись в файле `/etc/gshadow`. Для этого нужно вызвать `vigr` с ключом `-s` (`--shadow`):

```
sudo vigr -s
```

Чтобы получить дополнительную информацию об утилите `vigr`, обратитесь к соответствующей справочной странице (`man vigr`).

¹ В данном случае команда `getent` ничего не выводит, потому что группа `devops` больше не существует. — *Примеч. ред.*

Как и большинство задач в Linux, все предыдущие задачи можно было бы решить и другими способами. Мы перечислили команды, которые применяются чаще всего, но бывают случаи, когда лучше подходят другие средства.

В следующем разделе мы рассмотрим, как добавлять пользователей в основные и дополнительные группы и как удалять их оттуда.

Как регулировать членство пользователей в группах

До сих пор мы создавали только такие группы, в которых не было пользователей. Пустые группы не особо полезны, поэтому давайте добавим в них учетные записи.

Как добавить пользователей в группу

Прежде чем добавлять пользователей в группы, давайте создадим несколько групп. Для каждой группы мы запустим команду `groupadd` с ключом `-g (--gid)`, после которого укажем `GID`:

```
sudo groupadd -g 1100 admin
sudo groupadd -g 1200 developers
sudo groupadd -g 1300 devops
```

Вот как можно просмотреть последние созданные группы:

```
packt@neptune:~$ cat /etc/group | tail -n 5
alex:x:1002:
julian:x:1003:
admin:x:1100:
developers:x:1200:
devops:x:1300:
```

Эта команда выводит последние пять строк файла `/etc/group` — то есть пять последних групп, которые мы создали.

Далее давайте создадим две новые учетные записи (`alex2` и `julian2`, потому что пользователи `alex` и `julian` у нас уже есть) и добавим их в некоторые из свежих групп. Для обоих пользователей группа `admin` станет *основной (первичной)*, а группы `developers` и `devops` — *дополнительными (вторичными)*. Вот как можно создать пользователей и сразу же связать их с группами:

```
sudo useradd -g admin -G developers,devops alex2
sudo useradd -g admin -G developers,devops julian2
```

Ключ `-g (--gid)` команды `useradd` по-прежнему задает для каждой учетной записи единственную основную группу (в данном случае `admin`), а ключ `-G (--groups)` принимает список дополнительных групп, разделенных запятыми без пробелов (в данном случае — `developers,devops`).

Чтобы проверить членство в группах для каждого пользователя, можно запустить уже знакомую команду `id`:

```
packt@neptune:~$ id alex2
uid=1004(alex2) gid=1100(admin) groups=1100(admin),1200(developers),1300(devops)

packt@neptune:~$ id julian2
uid=1005(julian2) gid=1100(admin) groups=1100(admin),1200(developers),
1300(devops)
```

Как видите, атрибут `gid` показывает принадлежность к основной группе — в данном случае `gid=1100(admin)`, — а атрибут `groups` отражает дополнительные (вторичные) группы: `groups=1100(admin),1200(developers),1300(devops)`.

Когда пользователи распределены по нескольким группам, администратору иногда требуется перемещать учетные записи из одних групп в другие. В следующем разделе показано, как это сделать.

Как перемещать учетные записи между группами и исключать из групп

Продолжая предыдущий пример, предположим, что администратор хочет переместить (или добавить) пользователя `alex2` в новую вторичную группу `managers`. Обратите внимание, что в результате наших предыдущих упражнений пользователь `alex2` состоит в первичной группе `admin` и во вторичных группах `developers` и `devops` (см. вывод команды `id alex2` в предыдущем листинге).

Сначала создадим группу `managers` с GID 1400:

```
sudo groupadd -g 1400 managers
```

Далее добавим учетную запись `alex2` в группу `managers`. Напоминаем, что команда `usermod` с ключом `-G (--groups)` позволяет указать дополнительные (вторичные) группы, с которыми связан пользователь. Самый простой способ *добавить* вторичную группу — это использовать ключ `-a (--append)`:

```
sudo usermod -aG managers alex2
```

Эта команда сохранит существующие дополнительные группы для пользователя `alex2` и добавит его в новую группу `managers`. Вместо этого можно было запустить такую команду:

```
sudo usermod -G developers,devops,managers alex2
```

Обратите внимание, что здесь мы перечислили несколько групп через запятую (без пробелов!).

В предыдущем примере мы перечислили дополнительные группы `developers` и `devops`, в которых уже состоит пользователь `alex2`, и *добавили* в список еще одну дополнительную группу `managers`. Если бы мы указали только группу `managers`, то пользователь был бы *исключен* из групп `developers` и `devops`.

Можно убедиться, что пользователь alex2 теперь состоит в группе managers:

```
packt@neptune:~$ id alex2
uid=1004(alex2) gid=1100(admin)
groups=1100(admin),1200(developers),1300(devops),1400(managers)
```

Как видите, в атрибут groups входит запись для нужной группы: 1400(managers).

А если бы мы хотели *исключить* пользователя alex2 из вторичных групп developers и devops и связать его только со вторичной группой managers, то нужно было запустить такую команду:

```
packt@neptune:~$ sudo usermod -G managers alex2

packt@neptune:~$ id alex2
uid=1004(alex2) gid=1100(admin) groups=1100(admin),1400(managers)
```

Здесь в атрибуте groups отображается основная группа admin (по умолчанию) и дополнительная группа managers.

Чтобы исключить пользователя alex2 из всех вторичных групп, запустите такую команду:

```
sudo usermod -G '' alex2
```

Здесь команде usermod передается ключ -G (--groups) с пустой строкой (''), чтобы с пользователем не были связаны никакие дополнительные группы. Чтобы убедиться, что alex2 больше не состоит ни в одной вторичной группе, можно запустить такую команду:

```
packt@neptune:~$ id alex2
uid=1004(alex2) gid=1100(admin) groups=1100(admin)
```

Как видите, в атрибут groups входит только основной GID, который по умолчанию отображается для любого пользователя, — в данном случае gid=1100(admin).

Если администратор захочет удалить пользователя alex2 из основной группы или переместить его в другую основную группу, он может запустить команду usermod с ключом -g (--gid), после которого следует имя основной группы. В Linux каждый пользователь должен быть связан с основной группой, и она должна существовать.

Например, такая команда позволяет переместить пользователя alex2 в основную группу managers:

```
sudo usermod -g managers alex2
```

После этого с помощью id можно убедиться, что учетная запись перенесена в другую группу:

```
packt@neptune:~$ id alex2
uid=1004(alex2) gid=1100(admin) groups=1100(admin)

packt@neptune:~$ sudo usermod -g managers alex2

packt@neptune:~$ id alex2
uid=1004(alex2) gid=1400(managers) groups=1400(managers)
```

Атрибут `gid` в последней строке этого листинга отражает новую основную группу: `gid=1400(managers)`.

Если вы хотите, чтобы ни одна из уже существующих групп не служила основной для пользователя `alex2`, вам понадобится сначала создать эксклюзивную группу (для удобства назовем ее тоже `alex2`), `GID` которой совпадает с `UID` пользователя `alex2` (`1004`):

```
sudo groupadd -g 1004 alex2
```

После этого можно исключить пользователя `alex2` из текущей основной группы (`managers`), указав эксклюзивную основную группу, которую мы только что создали (`alex2`):

```
packt@neptune:~$ sudo usermod -g alex2 alex2

packt@neptune:~$ id alex2
uid=1004(alex2) gid=1004(alex2) groups=1004(alex2)
```

Здесь атрибут `gid` отражает эксклюзивную основную группу, имя и `GID` которой совпадают с именем и `UID` учетной записи соответственно: `gid=1004(alex2)`. Наш пользователь больше не состоит ни в каких других первичных группах.

Чем больше становится учетных записей и групп, тем сложнее администратору Linux контролировать членство пользователей в группах. Чтобы не только представлять себе реальную картину, но и эффективно автоматизировать работу пользователей, важно в любой момент знать, какие пользователи состоят в каких группах. В следующем разделе перечислены команды, которые позволяют просматривать сведения о пользователях и группах.

Как просматривать сведения о пользователях и группах

В этом разделе мы рассмотрим несколько полезных команд для того, чтобы получать сведения о группах и членстве в них. Прежде чем перейти к командам, напомним, что данные о группах хранятся в файлах `/etc/group` и `/etc/gshadow`, причем первый из них будет интересовать нас больше всего.

Вот как можно разобрать файл `/etc/group`, чтобы получить список всех групп:

```
packt@neptune:~$ cat /etc/group | cut -d: -f1 | column | less
root          dip          input
daemon        www-data     sgx
bin           backup       kvm
sys           operator     render
```

adm	list	lxd
tty	irc	_ssh
disk	src	crontab
lp	gnats	syslog
mail	shadow	uudd
newsa	utmp	tcpdump
uucp	video	tss
man	sasl	landscape
proxy	plugdev	packt
kmem	staff	fwupd-refresh
dialout	games	plocate
fax	users	alex
voice	nogroup	julian
cdrom	systemd-journal	admin
floppy	systemd-network	developers
tape	systemd-resolve	devops
sudo	messagebus	managers
audio	systemd-timesync	alex2

Такой же результат отобразит команда `getent`, если вызвать ее так:

```
getent group | cut -d: -f1 | column | less
```

Чтобы получить информацию об отдельной группе (например, `developers`), можно использовать такую команду:

```
packt@neptune:~$ getent group developers
developers:x:1200:julian2
```

Предыдущая команда также выводит список всех членов группы `developers` (в данном случае это пользователь `julian2`). Наоборот, чтобы узнать, в каких группах состоит тот или иной пользователь, можно запустить команду `groups`. Например, так выводится список всех групп, в которых состоит пользователь `alex`:

```
packt@neptune:~$ groups alex
alex : alex devops managers
```

Эта команда перечисляет группы пользователя `alex`, начиная с основной группы (`alex`).

С помощью той же утилиты `groups` пользователь может узнать, в каких группах он состоит, не указывая имя группы. Следующая команда выполняется в терминале пользователя `packt`, который также является администратором (супер-пользователем):

```
packt@neptune:~$ groups
packt adm cd rom sudo dip plugdev lxd
```

Существует много других способов и команд для того, чтобы получать сведения о пользователях и группах. Надеемся, что предыдущие примеры дают общее представление о том, где и как искать эти сведения.

Далее рассмотрим, как пользователи могут переключаться между группами или аутентифицироваться в определенных группах.

Групповые сеансы входа в систему

Когда пользователь входит в систему, контекст членства в группе автоматически устанавливается на его основную группу. В результате любая задача, которую он инициирует (например, если он создает файл или запускает программу), ассоциируется с правами доступа этой основной группы. Но пользователь также может получить доступ к ресурсам других групп, в которых он состоит (то есть дополнительных, или вторичных, групп). Чтобы переключить групповой контекст или войти в систему как член другой группы, можно вызвать утилиту командной строки `newgrp`:

```
newgrp имя_группы
```

В следующих примерах мы предполагаем, что пользователь `julian` состоит в нескольких группах: его основная группа — `admin`, а дополнительные — `developers` и `devops`:

```
packt@neptune:~$ id julian
uid=1003(julian) gid=1100(admin) groups=1100(admin),1200(developers),
1300(devops)
```

Давайте на некоторое время выдадим себя за пользователя `julian`. Сейчас мы аутентифицированы в системе как `packt`, а чтобы переключиться в учетную запись `julian`, выполним такую команду:

```
su julian
```

Обратите внимание, что если для пользователя `julian` не задан пароль, то войти в систему таким образом не получится.¹

Команда `whoami` выводит текущий контекст пользователя и группы:

```
packt@neptune:~$ whoami
packt

packt@neptune:~$ su julian
Password:

$ whoami
julian
```

Подробнее о команде `whoami` можно узнать с помощью `man whoami` или `whoami --help`.

¹ Чтобы вернуться из контекста другого пользователя в собственный, можно запустить команду `exit`. — *Примеч. ред.*

Команда `groups` выводит список групп, в которых состоит текущий пользователь (подробнее о ней можно узнать с помощью `man groups` или `groups --help`). Например, вот что выведет команда, если мы работаем под учетной записью `julian`:

```
$ groups
admin developers devops
```

А с помощью команды `id` пользователь может просмотреть свои идентификаторы (UID и GID):

```
$ id
uid=1003(julian) gid=1100(admin) groups=1100(admin),1200(developers),
1300(devops)
```

С помощью ключей команды `id` можно получить различные сведения о текущем пользователе и контексте группы. Например, ключ `-g (--group)` предоставляет GID группы, с которой связан текущий сеанс:

```
$ id -g
1100
```

В нашем случае эта команда выводит GID `1100`, который соответствует основной группе пользователя (`admin`). Когда вы входите в систему, то по умолчанию ваш сеанс ассоциируется с вашей основной группой. Например, если вы создадите файл, то права доступа к нему будут отражать идентификатор основной группы. Мы подробнее поговорим о правах доступа в разделе «Как управлять правами доступа».

Чтобы переключить групповой контекст для текущего пользователя на группу `developers`, можно запустить команду `newgrp`:

```
newgrp developers
```

После этого команда `id` будет отражать новую группу:

```
$ id -g
1200

$ id
uid=1003(julian) gid=1200(developers) groups=1200(developers),1100(admin),
1300(devops)
```

Здесь в атрибутах `gid` и `groups` фигурирует идентификатор вторичной группы `developers`. Если сейчас пользователь создаст какие-либо файлы, то права доступа к ним будут отражать GID группы `developers`.

Если пользователь пытается переключиться в группу, членом которой он не является (например, `managers`), то команда `newgrp` запрашивает пароль этой группы:

```
newgrp managers
Password:
```

Если пользователь знает пароль группы `managers` или является суперпользователем, он сможет успешно войти в группу, иначе ему будет отказано в доступе к ее ресурсам.

На этом мы завершаем тему управления пользователями и группами. В этом разделе мы постарались привести наиболее универсальные примеры соответствующих административных задач. Во многих случаях можно достичь одного и того же результата по-разному, применяя различные команды или приемы.

К этому моменту вы уже должны уверенно ориентироваться в том, как управлять пользователями и группами, а также владеть соответствующими утилитами командной строки. Между пользователями и группами существуют реляционные отношения: каждому пользователю соответствует одна или более групп. Не забывайте, что для того, чтобы создавать пользователей и группы и управлять ими, нужны полномочия суперпользователя. В Linux сведения о пользователях хранятся в файлах `/etc/passwd` и `/etc/shadow`, а о группах — в файлах `/etc/group` и `/etc/gshadow`. Чтобы модифицировать учетные записи и группы, можно использовать специальные утилиты командной строки, но можно и редактировать эти файлы вручную.

Далее мы рассмотрим контекст безопасности и изоляции в многопользовательской групповой среде. В Linux соответствующую функциональность обеспечивают права доступа, которые регулируются на уровне системы и контролируют пользовательские и групповые разрешения на чтение, запись и выполнение файлов и каталогов.

В следующем разделе рассматриваются задачи управления и администрирования, связанные с этими правами доступа.

Как управлять правами доступа

Один из ключевых принципов Linux состоит в том, что система может предоставлять доступ многим пользователям, которые одновременно выполняют независимые задачи. Чтобы эта многопользовательская и многозадачная среда работала бесперебойно, применяются **права доступа (разрешения)**. Ядро Linux обеспечивает надежную основу для базовой модели безопасности и изоляции, а на пользовательском уровне специальные инструменты и утилиты командной строки помогают пользователям и администраторам Linux решать соответствующие задачи управления доступом.

Некоторые пользователи Linux, особенно начинающие, могут путаться в том, как устроены разрешения. В этом разделе мы разъясним важнейшие принципы, на которых основаны права доступа к файлам и каталогам в Linux. Вы узнаете о том, каковы ключевые атрибуты прав — *чтение, запись и выполнение*,

а также освоите основные административные задачи: научитесь просматривать и изменять права доступа, используя утилиты командной строки системного уровня.

Большинство тем этого раздела стоит рассматривать в тесной связи с пользователями и группами. Соответствующие идиомы выражаются довольно просто, например: «пользователь может читать или обновлять такой-то файл», «у группы есть доступ к таким-то файлам и каталогам» или «пользователь может выполнять такую-то программу».

Давайте начнем с основ и познакомимся с правами доступа к файлам и каталогам.

Права доступа к файлам и каталогам

В Linux права доступа можно рассматривать как полномочия на то, чтобы выполнять те или иные операции с файлом или каталогом. Перечислим основные *атрибуты прав доступа*:

- **Чтение** (*read*). Разрешение на чтение файла или каталога позволяет просматривать его содержимое.
- **Запись** (*write*). Разрешение на запись файла позволяет изменять его содержимое. Для каталога это разрешение позволяет изменять содержимое каталога: добавлять в него файлы, удалять или переименовывать их.
- **Выполнение** (*execute*). Разрешение на выполнение файла позволяет запускать сценарий, приложение или службу, назначенную файлом. Для каталога разрешение на выполнение позволяет войти в каталог и сделать его текущим рабочим каталогом (с помощью команды `cd`).

Для начала давайте рассмотрим, как устроены права доступа к файлам и каталогам.

Как просматривать сведения о правах доступа

Самый распространенный способ просмотреть права доступа к файлу или каталогу — использовать команду `ls`. Ее синтаксис таков:

```
ls [ключи] имя_файла_или_каталога
```

Например, так можно просмотреть права доступа к файлу `/etc/passwd`:

```
packt@neptune:~$ ls -l /etc/passwd
-rw-r--r-- 1 root root 1476 Feb 12 16:57 /etc/passwd
```

Напоминаем, что команда `ls` с ключом `-l` выводит данные в расширенном формате (подробнее см. `man ls`). Этот формат состоит из девяти полей, разделенных пробелами:

-rw-r--r--	Тип файла и права доступа к нему
1	Количество жестких ссылок на файл
root	Пользователь, который владеет файлом
root	Группа, которая владеет файлом
2010	Размер файла
Feb 12 08:57	Дата и время создания файла
/etc/passwd	Имя файла

Рассмотрим подробнее права доступа к файлу (в данном случае -rw-r--r--). Как уже говорилось в разделе «Владельцы файлов и права доступа к файлам» (глава 2), первое поле состоит из 10 символов, первый из которых обозначает тип файла, а остальные девять — права доступа. Повторим, что эти права делятся на три блока по три символа: первый блок обозначает права владельца, второй — права группы-владельца, а последний — права остальных пользователей («всего мира»).

Каждую триаду можно представить либо как последовательность символов (например, rw-), либо как эквивалентное восьмеричное число (например, 6). Вот как эти представления соответствуют друг другу:

Восьмеричное число	Символьное обозначение	Права доступа
7	rwX	Чтение, запись и выполнение
6	rw-	Чтение и запись
5	r-w	Чтение и выполнение
4	r--	Только чтение
3	-wX	Запись и выполнение
2	-w-	Только запись
1	--X	Только выполнение
0	---	Нет доступа

Рассмотрим несколько примеров.

Примеры прав доступа к файлам

Некоторые примеры прав доступа уже встречались нам в разделе «Владельцы файлов и права доступа к файлам» (глава 2). Давайте закрепим знания на примере файла `/etc/passwd` и посмотрим, что означают его разрешения `rw- r-- r--:`

Триада	Обозначение	Субъект прав	Права	Числовое значение
1	rw-	Пользователь-владелец	Чтение (r) и запись (w)	4 + 2 = 6
2	r--	Группа-владелец	Только чтение (r)	4
3	r--	Остальные пользователи	Только чтение (r)	4

Таким образом, правам доступа к файлу `/etc/passwd` соответствует восьмеричное значение `644`. Его можно также запросить с помощью команды `stat`:

```
packt@neptune:~$ stat --format '%a' /etc/passwd
644
```

Команда `stat` отображает сведения о файле или файловой системе. Ключ `-c (--format)` позволяет вывести отдельные поля этих сведений: например, спецификатор `'%a'` обозначает права доступа в восьмеричном формате.

Вот несколько примеров прав доступа с соответствующими восьмеричными значениями и описаниями. Для наглядности блоки из трех символов разделены пробелами, а тип файла опущен:

Права доступа	Числовое значение	Описание
rwX rwX rwX	777	Все пользователи (владелец, группа-владелец и все остальные) могут читать, записывать и выполнять
rwX r-X r-X	755	Все пользователи могут читать и выполнять, а владелец также может записывать
rwX r-X ---	750	Владелец и группа-владелец могут читать и выполнять; владелец также может записывать; у остальных пользователей нет доступа
rwX --- ---	700	Владелец может читать, записывать и выполнять, а у всех остальных нет доступа
rw- rw- rw-	666	Все пользователи могут читать и записывать, никто не может выполнять
rw- rw- r--	664	Владелец и группа могут читать и записывать, все остальные — только читать
rw- rw- ---	660	Владелец и группа могут читать и записывать, а у всех остальных нет доступа
rw- r-- r--	644	Владелец может читать и записывать, а все остальные — только читать

Права доступа	Числовое значение	Описание
<code>rw- r-- ---</code>	640	Владелец может читать и записывать, группа-владелец — только читать, а у всех остальных нет доступа
<code>rw- --- ---</code>	600	Только владелец может читать и записывать, а у всех остальных нет доступа
<code>r-- --- ---</code>	400	Владелец может только читать, а у всех остальных нет доступа

Чтение, запись и выполнение — наиболее распространенные атрибуты прав доступа к файлам. До сих пор речь шла в основном о том, какие бывают права доступа и как они обозначаются. В следующем разделе мы рассмотрим несколько инструментов командной строки, с помощью которых можно регулировать права доступа.

Как регулировать права доступа

Настраивать разрешения доступа к файлам и каталогам — распространенная задача администратора Linux. В этом разделе мы познакомимся с несколькими утилитами командной строки, которые позволяют изменять права доступа и владельцев файлов и каталогов. Эти утилиты поставляются со всеми современными дистрибутивами Linux и в большинстве дистрибутивов используются одинаково.

Команда `chmod` — редактирование прав доступа

Название команды `chmod` — сокращение от *change mode*¹, и она предназначена для того, чтобы задавать права доступа к файлам и каталогам. Команду `chmod` может запускать как суперпользователь, так и текущий пользователь (для файлов, которыми он владеет).

Права доступа можно настраивать в двух режимах: **относительном** и **абсолютном**. Давайте рассмотрим каждый из них.

Использование `chmod` в относительном режиме

Редактировать права в **относительном** режиме очень просто, важно лишь указать подробности по такой схеме:

Чьи права доступа вы изменяете	u	Владелец
	g	Группа-владелец
	o	Все остальные пользователи

¹ Изменить режим. — Примеч. пер.

Как именно вы изменяете права доступа	+	Добавить право доступа
	-	Удалить право доступа
	=	Установить права в точности с указанными
Какие именно права доступа вы изменяете	r	Чтение
	w	Запись
	x	Выполнение

Рассмотрим несколько примеров того, как можно использовать `chmod` в относительном режиме.

В первом примере мы хотим добавить (+) права на запись (w) в файл `myfile` для всех *остальных* пользователей (o) — то есть для тех, кто не является владельцем файла и не состоит в группе-владельце:

```
chmod o+w myfile
```

Так выглядит соответствующий вывод в терминале:

```
packt@neptune:~$ touch myfile
packt@neptune:~$ ls -l myfile
-rw-rw-r-- 1 packt packt 0 Mar  9 17:12 myfile
packt@neptune:~$ chmod o+w myfile
packt@neptune:~$ ls -l myfile
-rw-rw-rw- 1 packt packt 0 Mar  9 17:12 myfile
```

В следующем примере мы удаляем (-) права на чтение (r) и запись (w) в том же файле для текущего пользователя-владельца (u):

```
chmod u-rw myfile
```

Так выглядит соответствующий вывод в терминале:

```
packt@neptune:~$ ls -l myfile
-rw-rw-rw- 1 packt packt 0 Mar  9 17:12 myfile
packt@neptune:~$ chmod u-rw myfile
packt@neptune:~$ ls -l myfile
---rw-rw- 1 packt packt 0 Mar  9 17:12 myfile
```

В предыдущих примерах мы не использовали `sudo`, потому что выполняли операции от имени текущего владельца файла (`packt`).

В следующем примере мы предполагаем, что файл `myfile` доступен всем пользователям для чтения, записи и выполнения, и хотим внести такие изменения:

- Удалить (-) право на чтение (r) для владельца (u)
- Удалить (-) право на запись (w) для владельца (u) и группы (g)
- Удалить (-) права на чтение (r), запись (w) и выполнение (x) для всех остальных (o)

Перечислим все три изменения в одной команде через запятую:

```
chmod u-r,ug-w,o-rwx myfile
```

Так выглядит соответствующий вывод в терминале:

```
packt@neptune:~$ ls -l myfile
-rwxrwxrwx 1 packt packt 0 Mar  9 17:12 myfile

packt@neptune:~$ chmod u-r,ug-w,o-rwx myfile

packt@neptune:~$ ls -l myfile
---xr-x--- 1 packt packt 0 Mar  9 17:12 myfile
```

Теперь рассмотрим второй способ изменения прав доступа: мы будем использовать утилиту `chmod` в абсолютном режиме, указывая восьмеричное число, которое соответствует нужным правам.

Использование `chmod` в абсолютном режиме

Когда команда `chmod` вызывается в **абсолютном** режиме, она изменяет сразу все атрибуты прав доступа в соответствии с *восьмеричным* значением, которое передано команде. Этот режим называется абсолютным, потому что он изменяет все права без привязки к тем, которые действовали ранее.

Таблица числовых значений прав доступа и соответствующие примеры уже приводились в разделе «Как просматривать сведения о правах доступа» ранее в этой главе. А в следующем примере мы сделаем так, чтобы все пользователи могли читать (r), записывать (w) и выполнять (x) файл `myfile`:

```
packt@neptune:~$ chmod 777 myfile

packt@neptune:~$ ls -l myfile
-rwxrwxrwx 1 packt packt 0 Mar  9 17:12 myfile
```

Чтобы получить дополнительную информацию о команде `chmod`, обратитесь к соответствующей справочной странице (`man chmod`).

Теперь рассмотрим еще одну утилиту командной строки, с помощью которой можно изменять владельцев файлов и каталогов.

Команда `chown` — назначение владельцев файлов

Команда `chown` (сокращение от *change owner*¹) позволяет назначать владельцев файлов и каталогов. Как правило, ее можно запускать только с полномочиями

¹ Изменить владельца. — *Примеч. пер.*

суперпользователя. Обычные пользователи могут изменять только *групповую* принадлежность своих файлов, причем только если эти пользователи состоят в целевой группе.

Синтаксис команды `chown` таков:

```
chown [ключи] [владелец][:[группа]] имя_файла
```

Когда вызывают команду `chown`, чаще всего указывают как пользователя, так и группу, например:

```
packt@neptune:~$ ls -l myfile
-rwxrwxrwx 1 packt packt 0 Mar  9 17:12 myfile

packt@neptune:~$ sudo chown julian:developers myfile
[sudo] password for packt:

packt@neptune:~$ ls -l myfile
-rwxrwxrwx 1 julian developers 0 Mar  9 17:12 myfile
```

Команду `chown` очень часто запускают в *рекурсивном режиме* с ключом `-R` (`--recursive`). В следующем примере мы назначаем пользователя `julian` владельцем каталога `mydir` вместе со всеми вложенными файлами и каталогами:

```
sudo chown -R julian:julian mydir/
```

Обратите внимание, какой пользователь и какая группа значатся в качестве владельцев до того, как мы вызвали эту команду, а какие — после этого:

```
packt@neptune:~$ ls -lR mydir/
mydir/:
total 4
-rw-rw-r-- 1 packt packt  0 Mar  9 17:35 file1
-rw-rw-r-- 1 packt packt  0 Mar  9 17:35 file2
drwxrwxr-x 2 packt packt 4096 Mar  9 17:37 subdir

mydir/subdir:
total 0
-rw-rw-r-- 1 packt packt 0 Mar  9 17:37 file1
-rw-rw-r-- 1 packt packt 0 Mar  9 17:37 file2

packt@neptune:~$ sudo chown -R julian:julian mydir/

packt@neptune:~$ ls -lR mydir/
mydir/:
total 4
-rw-rw-r-- 1 julian julian  0 Mar  9 17:35 file1
-rw-rw-r-- 1 julian julian  0 Mar  9 17:35 file2
drwxrwxr-x 2 julian julian 4096 Mar  9 17:37 subdir

mydir/subdir:
total 0
-rw-rw-r-- 1 julian julian 0 Mar  9 17:37 file1
-rw-rw-r-- 1 julian julian 0 Mar  9 17:37 file2
```

Чтобы получить дополнительную информацию о команде `chown`, обратитесь к соответствующей справочной странице (`man chown`).

Далее пойдет речь о похожей утилите командной строки, которая назначает только группы-владельцы.

Команда `chgrp` — назначение групп-владельцев

Команда `chgrp` (сокращение от *change group*¹) позволяет назначить *группу-владельца* файлов и каталогов. Напомним, что с файлами и каталогами в Linux обычно связаны пользователь-владелец и группа-владелец. Пользователя можно назначить владельцем с помощью утилиты `chown`, а группу — также с помощью `chgrp`. Синтаксис этой команды таков:

```
chgrp [ключи] группа имя_файла
```

В следующем примере мы назначаем группу `developers` группой-владельцем файла `myfile`:

```
packt@neptune:~$ sudo chgrp developers myfile
packt@neptune:~$ ls -l myfile
-rwxrwxrwx 1 julian developers 0 Mar  9 17:12 myfile
```

Мы вызвали предыдущую команду с полномочиями суперпользователя (`sudo`), потому что текущий пользователь `packt` не является администратором группы `developers`.

Чтобы получить дополнительную информацию об утилите `chgrp`, обратитесь к ее встроенной справке (`chgrp --help`).

Команда `umask` — права доступа по умолчанию

Команда `umask` позволяет просмотреть или задать права доступа, применяющиеся по умолчанию к новым файлам и каталогам, которые создает пользователь в системе. Значение `umask` формально называется *маской создания файлового режима*. Например, в Ubuntu по умолчанию применяются такие маски:

- `0002` — для обычных пользователей;
- `0022` — для пользователя `root`.

Как правило, в Linux *права доступа по умолчанию* для новых файлов и каталогов рассчитываются по таким формулам²:

¹ Изменить группу. — *Примеч. пер.*

² Строго говоря, окончательные права доступа к файлу или каталогу задаются как $x \wedge \overline{m}$, где x — базовые права процесса, который создает файл или каталог (например, `0666`), $\overline{m}$ — логическая инверсия маски, а $\wedge$ — логическое сложение (оператор И). Маску создания файлового режима также можно понимать как перечень прав доступа, которые мы не хотим назначить файлу. — *Примеч. ред.*

Для новых файлов	0666 - umask
Для новых каталогов	0777 - umask

Из этого следует, что в Ubuntu применяются такие права доступа по умолчанию:

	Обычные пользователи	root
Файлы	0666 - 0002 = 0664	0666 - 0022 = 0644
Каталоги	0777 - 0002 = 0775	0777 - 0022 = 0755

В следующем упражнении на примере Ubuntu мы создаем файл `myfile` и каталог `mydir` в контексте обычного пользователя (`packt`). Затем мы запускаем команду `stat` для каждого из них и убеждаемся, что права доступа по умолчанию соответствуют перечисленным значениям для обычных пользователей (664 для файла и 775 для каталога):

```
packt@neptune:~$ touch myfile2
packt@neptune:~$ stat -c '%a' myfile2
664

packt@neptune:~$ mkdir mydir2
packt@neptune:~$ stat -c '%a' mydir2
775
```

Вот типичные значения `umask` и соответствующие им права доступа для файлов и каталогов в системах Linux:

umask	Файлы	Каталоги
0000	666 rw-rw-rw-	777 rwxrwxrwx
0002	664 rw-rw-r--	775 rwxrwxr-x
0022	644 rw-r--r--	755 rwxr-xr-x
0027	640 rw-r-----	750 rwxr-x---
0077	600 rw-----	700 rwx-----
0277	400 r-----	500 r-x-----

Чтобы получить дополнительную информацию об утилите `umask`, обратитесь к ее встроенной справке (`umask --help`).

Права доступа к файлам и каталогам играют важнейшую роль в модели безопасности операционной системы. Пользователи и процессы должны работать в изолированном окружении и в рамках ограничений, которые накладывают права доступа, чтобы никто не мог случайно или преднамеренно нанести ущерб, вмешавшись в управление системными ресурсами. Бывают случаи, когда в права доступа входят некоторые специальные атрибуты; они особенно востребованы, если вы работаете от имени другого пользователя. Давайте рассмотрим эти атрибуты.

Специальные атрибуты прав доступа

В Linux владельцы файлов и каталогов обычно назначаются в соответствии с UID и GID пользователя или группы, которые их создали. Тот же принцип применяется к приложениям и процессам: их владельцами считаются пользователи, которые их запускают. Специальные атрибуты прав доступа позволяют изменять это поведение по умолчанию, когда нужно.

Вот флаги специальных атрибутов и соответствующие восьмеричные значения:

Флаг	Номер установленного бита	Числовое значение
setuid	2	$2^2 = 4$
setgid	1	$2^1 = 2$
sticky	0	$2^0 = 1$

Когда какой-либо из этих специальных битов установлен, итоговое восьмеричное значение прав доступа состоит не из трех, а из четырех цифр, причем первая из них соответствует восьмеричному значению специальных атрибутов.

Давайте рассмотрим каждый из этих атрибутов и приведем практические примеры.

Флаг setuid

Если для исполняемого файла установлен флаг **setuid**, то этот файл будет выполняться с правами своего владельца, а не того пользователя, который его запустил. Например, если владелец исполняемого файла — **root**, а запускает его *обычный* пользователь, то соответствующий процесс будет выполняться с привилегиями **root**. Бит **setuid** может представлять потенциальную угрозу безопасности, если использовать его неадекватно или если злоумышленнику удастся эксплуатировать уязвимости соответствующего процесса.

В поле прав доступа к исполняемому файлу бит **setuid** представляется строчной буквой **s**, которая *заменяет* соответствующую букву **x** (право на выполнение файла). Для неисполняемого файла используется прописная буква **S**.

Флаг **setuid** можно установить с помощью команды **chmod**. Например, так это делается для исполняемого файла **myscript.sh**:

```
packt@neptune:~$ ls -l myscript.sh
-rwxrw-r-- 1 packt packt 0 Feb 15 16:56 myscript.sh

packt@neptune:~$ chmod u+s myscript.sh

packt@neptune:~$ ls -l myscript.sh
-rwsrw-r-- 1 packt packt 0 Feb 15 16:56 myscript.sh

packt@neptune:~$ stat -c '%a' myscript.sh
4744
```

Обратите внимание на то, как изменились права доступа. До того, как мы применили `chmod`, соответствующее поле выглядело как `-rwxrwxr-x`, а после этой команды в нем появилась буква `s`, которая соответствует флагу `setuid`: `-rwsrwxr-x`. Дополнительные сведения о `setuid` можно найти на справочной странице команды `chmod` (`man chmod`).

Флаг `setgid`

В то время как `setuid` влияет на права доступа пользователей, `setgid` аналогичным образом регулирует разрешения групп.

Если для исполняемого файла установлен флаг `setgid`, то этот файл будет выполняться с правами той группы, которая владеет файлом, а не группы того пользователя, который его запустил. Другими словами, GID соответствующего процесса совпадает с GID файла.

Если флаг `setgid` применяется к каталогу, это приводит к тому, что группой-владельцем файлов, которые создаются в каталоге, назначается группа-владелец родительского каталога, а не группа пользователя, который их создает. Это бывает уместно, когда при совместном использовании файлов нужно, чтобы их могли изменять все пользователи, которые связаны с группой-владельцем родительского каталога.

Флаг `setgid` также можно установить с помощью `chmod`. Например, зададим его для исполняемого файла `myscript.sh` (допустим, что это исходная версия файла, к которой еще не применяли `setuid`):

```
packt@neptune:~$ chmod g+s myscript.sh

packt@neptune:~$ ls -l myscript.sh
-rwxrwsr-- 1 packt packt 0 Feb 15 16:56 myscript.sh

packt@neptune:~$ stat -c '%a' myscript.sh
2775
```

Чтобы получить дополнительную информацию о `setgid`, посетите страницу en.wikipedia.org/wiki/Setuid или обратитесь к справочной странице `chmod` (`man chmod`).

Флаг `sticky`

Флаг `sticky` (он же `sticky`-бит, буквально «липкий бит») не применяется к обычным файлам.¹ Если он задан для каталога, то удалять или переименовывать файлы в этом каталоге могут только его владельцы (пользователь-владелец или

¹ Sticky-бит особым образом применялся к исполняемым файлам в ранних версиях Unix и до сих пор доступен в некоторых системах семейства Unix. В Linux этот флаг технически можно задать для обычного файла (в выводе `ls` он будет обозначаться прописной буквой `T`), но это не вызовет никаких специальных эффектов. — *Примеч. ред.*

члены группы-владельца), а другие пользователи, у которых есть право на запись в этот каталог, не могут выполнять эти операции. Sticky-бит полезен, когда каталогом владеет специально назначенная группа, членам которой нужны общие права на запись файлов в этом каталоге.

Флаг `sticky` можно установить с помощью такой команды `chmod` (например, для каталога `mydir`):

```
packt@neptune:~$ mkdir mydir
packt@neptune:~$ chmod +t mydir

packt@neptune:~$ ls -ld mydir
drwxrwxr-t 2 packt packt 4096 Mar  9 17:10 mydir

packt@neptune:~$ stat -c '%a' mydir
1775
```

Как видите, права доступа к каталогу имеют вид `drwxrwxr-t`, а соответствующее восьмеричное значение равно `1775`.

Чтобы получить дополнительную информацию о флаге `sticky`, посетите страницу en.wikipedia.org/wiki/Setuid или обратитесь к справочной странице `chmod` (`man chmod`).

Иногда бывает непросто разобраться в правах доступа к файлам и каталогам. Надеемся, что этот раздел помог вам понять некоторые тонкости, связанные с разрешениями, и что теперь вы сможете чувствовать себя уверенее, когда будете встречаться с правами доступа в повседневных задачах администрирования Linux.

Итоги

В этой главе мы познакомились с основными понятиями, которые относятся к управлению пользователями и группами в Linux. Вы узнали о правах доступа к файлам и каталогам и о различных уровнях доступа в многопользовательской среде. По каждой основной теме мы рассмотрели основные административные задачи и привели практические примеры того, как с помощью типичных инструментов командной строки выполнять повседневные операции по управлению разрешениями.

Всякому администратору Linux важно уметь управлять пользователями и группами, а также настраивать соответствующие права доступа к ресурсам файловой системы. Мы надеемся, что знания, которые вы получили в этой главе, помогут вам стать опытными суперпользователями.

В следующей главе мы продолжим знакомиться с внутренним устройством Linux и изучим процессы, демоны и механизмы **межпроцессного взаимодействия (ИПС)**. Важно помнить, что у процессов и демонов тоже есть владельцы — пользователи или группы. Все, что вы узнали в этой главе, поможет вам ориен-

тироваться в новой теме, когда мы будем выяснять, *кто что запускает* в системе в тот или иной момент времени.

Вопросы

Вот несколько контрольных вопросов, которые помогут закрепить материал этой главы:

1. Что такое суперпользователь?

Подсказка: попробуйте применить `sudo`.

2. Какая утилита командной строки позволяет создавать пользователей? А можете ли вы назвать еще одну утилиту?

Подсказка: вспомните про `adduser` и `useradd`.

3. Какое восьмеричное значение соответствует правам доступа `-rw-rw-r-`?

Подсказка: вспомните, что атрибуты `r`, `w` и `x` представляются как восьмеричные числа `4`, `2` и `1` соответственно.

4. Чем основная (первичная) группа отличается от дополнительной (вторичной)?

5. Как изменить владельца домашнего каталога пользователя?

6. Можно ли удалить пользователя из системы, не удаляя его домашний каталог? Как это сделать?

Дополнительная литература

- Джей Лакруа (Jay LaCroix) «Mastering Ubuntu Server», Fourth Edition.
- Пабло Иранзо Гомес и др. (Pablo Iranzo Gomez et al.) «Red Hat Enterprise Linux 9 Administration», Second Edition.

5

Как работать с процессами, демонами и сигналами

Linux — многозадачная операционная система. Несколько программ или задач могут выполняться параллельно, и у каждой будет свой идентификатор, планирование, выделенная память, права доступа и системные ресурсы. Контексту выполнения любой такой программы соответствует абстракция **процесса**. Любому системному администратору или разработчику, который имеет дело с Linux, крайне важно понимать, как устроены процессы и как они взаимодействуют друг с другом.

Эта глава посвящена основным понятиям процессов в Linux. Мы рассмотрим различные типы процессов — такие, как процессы **переднего плана** и **фоновые** процессы, и уделим особое внимание **демонам** — особому виду фоновых процессов. Мы внимательно изучим внутреннее устройство процессов и различные механизмы межпроцессного взаимодействия в Linux — в частности, **сигналы**. Попутно мы познакомимся с некоторыми важными утилитами командной строки, которые позволяют управлять процессами и демонами, а также работать с сигналами. Кроме того, впервые в этой книге вы встретитесь со **сценариями**, о которых мы подробнее поговорим в главе 8 «Сценарии командной оболочки Linux». Если по ходу текущей главы вы почувствуете, что вам нужна дополнительная информация о сценариях, ознакомьтесь с главой 8.

В этой главе мы рассмотрим такие темы:

- Что такое процессы.
- Как работать с процессами.
- Как работать с демонами.
- Межпроцессное взаимодействие.

Далее мы будем время от времени упоминать сигналы *до того*, как полноценно представим их во второй половине этой главы. В Linux сигналы практически нигде не фигурируют отдельно от процессов, поэтому мы решили сначала познакомить вас с процессами. Тем не менее, если при этом вообще не затрагивать тему сигналов, это помешает понять некоторые особенности внутреннего устройства процессов. Каждый раз, когда упоминаются сигналы, мы будем ссылаться на раздел, в котором нужная тема рассматривается подробнее. Думаем, что такой подход поможет вам получить общее представление о процессах и демонах и понять, как они устроены.

Прежде чем начать, давайте рассмотрим технические требования.

Технические требования

Как известно, теория без практики мертва. Мы уверены, что если вы будете вручную запускать команды и воспроизводить примеры из этой главы, то сможете ориентироваться в процессах на профессиональном уровне. Как и в любой другой главе этой книги, мы рекомендуем, чтобы у вас был действующий дистрибутив Linux, который установлен на виртуальной машине или настольном компьютере. Здесь мы будем использовать Ubuntu или Fedora, но большинство команд и примеров можно воспроизвести и на любой другой платформе Linux.

Что такое процессы

Процесс — это, по сути, запущенный экземпляр программы. В общем случае программа — это совокупность инструкций и данных, которая скомпилирована в виде исполняемого кода. Когда программа запускается, возникает соответствующий процесс. Другими словами, процесс — это просто программа в действии. Каждый процесс выполняет определенные задачи, так что процессы иногда также называют **заданиями** (или **задачами**).

Создать или запустить процесс можно многими способами. В Linux каждая команда запускает процесс. В качестве команды может выступать задача, которую инициирует пользователь в командной строке, или сценарий либо программа (исполняемый файл), которые запускают ручную или автоматически.

Обычно тип процесса определяется тем, как он создается и взаимодействует с системой (или пользователем). Давайте подробнее рассмотрим различные типы процессов в Linux.

Типы процессов в Linux

«С высоты птичьего полета» в Linux можно выделить два основных типа процессов:

- **процессы переднего плана** (интерактивные);
- **фоновые процессы** (неинтерактивные, или автоматические).

Интерактивные процессы так или иначе взаимодействуют с пользователем, а неинтерактивные не требуют обслуживания: они либо запускаются автоматически (например, при загрузке системы), либо планируются на определенный день и время с помощью планировщиков заданий (например, таких, как утилиты `at` и `cron`).

Рассматривая типы процессов, мы будем в основном опираться на это разделение. Процессы можно классифицировать и другими способами, но в конечном

итоге все сводится к тому, что одни процессы выполняются на переднем плане, а другие — в фоне.

Например, пакетные процессы и демоны по сути относятся к фоновым. Пакетные процессы автоматизированы в том смысле, что их инициирует не пользователь, а задача планировщика. А демоны — это фоновые процессы, которые обычно запускаются во время загрузки системы и работают в постоянном режиме.

Между процессами также существуют иерархические отношения. Если процесс создает новые подчиненные процессы, то он называется родительским по отношению к ним, а эти процессы — дочерними по отношению к нему.

Мы подробно рассмотрим эти отношения (и не только) в следующих разделах. Начнем с основных типов — процессов переднего плана и фоновых процессов.

Процессы переднего плана

Процессы переднего плана, которые также называются **интерактивными**, порождаются и управляются через сеанс терминала. Обычно их запускает пользователь с помощью интерактивного интерфейса командной строки. Процесс переднего плана может выводить результаты в консоль (`stdout` или `stderr`) или принимать ввод от пользователя. Время жизни процесса переднего плана тесно связано с сеансом терминала (родительским процессом). Если пользователь, который запустил процесс переднего плана, выйдет из терминала, пока процесс еще выполняется, то этот процесс принудительно завершится (по сигналу `SIGHUP` от родительского процесса; подробнее см. «Сигналы в Linux» в разделе «Межпроцессное взаимодействие»).

Простой пример процесса переднего плана — вызов справочной страницы (`man`) для произвольной команды Linux (допустим, `ps`):

```
man ps
```

Команда `ps` выводит сведения об активных процессах. Подробнее об инструментах управления процессами и соответствующих утилитах командной строки вы узнаете в разделе «Как работать с процессами».

После того как запущен процесс переднего плана, его интерфейс перехватывает и контролирует приглашение командной строки. Пользователь больше не может взаимодействовать с исходной командной строкой, пока интерактивный процесс не передаст управление сеансу терминала.

Давайте рассмотрим еще один пример процесса переднего плана: на этот раз он вызывает долговременную задачу. Следующая однострочная команда запускает бесконечный цикл, который каждые несколько секунд выводит на экран сообщение:

```
while true; do echo "Ожидайте..."; sleep 5; done
```


Если запустить эту команду и не прерывать ее принудительно, она будет выводить сообщения неограниченно долго, и у пользователя не будет доступа к интерактивному приглашению командной строки. Если в терминале нажать **Ctrl + C**, процесс переднего плана будет прерван и мы вернемся в командную строку:

```
packt@neptune:~$ while true; do echo "Ожидайте..."; sleep 5; done
Ожидайте...
Ожидайте...
Ожидайте...
^C

packt@neptune:~$ _
```

Если нажать **Ctrl + C**, когда выполняется процесс переднего плана, то текущий (родительский) сеанс терминала передает этому процессу сигнал **SIGINT** и процесс прерывается. Дополнительные сведения см. в разделе «Сигналы в Linux».

Если вы хотите, чтобы в сеансе терминала оставалась интерактивная командная строка, пока выполняется та или иная команда или сценарий, используйте фоновый процесс.

Фоновые процессы

Фоновые процессы, которые также называются **неинтерактивными**, или **автоматическими**, выполняются независимо от сеанса терминала и не предполагают никакого взаимодействия с пользователем. В рамках одного терминального сеанса пользователь может запустить несколько фоновых процессов, не дожидаясь, пока каждый из них завершится.

Фоновые процессы — это обычно долговременные задачи, которые пользователю не нужно непосредственно контролировать. В принципе, соответствующий процесс может по-прежнему выводить свои результаты в консоль терминала, но обычно такие фоновые задачи записывают вывод в другие места (например, в файлы журнала).

Чтобы запустить фоновый процесс, можно просто добавить амперсанд (**&**) в конец соответствующей команды. Преобразуем предыдущий пример из раздела «Процессы переднего плана» так, чтобы породить фоновый процесс, который запускает бесконечный цикл и выводит сообщение каждые несколько секунд:

```
while true; do echo "Ожидайте..."; sleep 10; done &
```

Обратите внимание на амперсанд (**&**) в конце команды. Благодаря ему сеанс терминала остается интерактивным, хотя по умолчанию фоновый процесс по-

прежнему направляет вывод (`stdout` и `stderr`) в консоль. В следующем примере мы запускаем команду `echo`, пока предыдущий процесс еще выполняется:

```
packt@neptune:~$ while true; do echo "Ожидайте..."; sleep 10; done &
[1] 983
packt@neptune:~$ Ожидайте...
echo "Интерактивный Ожидайте...
режим"
Интерактивный режим
packt@neptune:~$ Ожидайте...
kill -9 983
packt@neptune:~$ _
```

Как вы наверняка заметили, фоновому процессу присвоен **идентификатор процесса (PID) 983**. Хотя процесс выполняется, это не мешает нам управлять сеансом терминала и запустить другую команду, например:

```
echo "Интерактивный режим"
```

В конце концов мы принудительно завершаем процесс с помощью команды `kill`:

```
kill -9 983
```

Эта команда «убивает» процесс с PID 983 (то есть наш фоновый процесс). Для этого родительский сеанс терминала передает ему сигнал `SIGKILL` (см. раздел «Сигналы в Linux»), потому что мы указали ключ `-9`.

Пользователь обычно может непосредственно контролировать как процессы переднего плана, так и фоновые процессы. Другими словами, и те и другие создаются или запускаются вручную, когда вы вводите команду или вызываете сценарий. Из этого правила бывают исключения, особенно когда речь идет о пакетных процессах, которые запускаются автоматически с помощью запланированных заданий.

Есть еще одна категория фоновых процессов, которые автоматически запускаются при загрузке системы и завершают работу вместе с ней без контроля со стороны пользователя. Эти фоновые процессы также известны как демоны.

Что такое демоны

Демон — это особый тип фонового процесса, который обычно запускается при загрузке системы и работает неопределенно долго (или пока система не выключится). У демона нет терминала, через который пользователь может им управлять, хотя демон связан с системной учетной записью (`root` или другой) и запускается с соответствующими правами.

Обычно демоны обслуживают запросы клиентов или взаимодействуют с другими процессами (фоновыми или переднего плана). Вот типичные примеры демонов, которые обычно доступны на большинстве платформ Linux:

systemd	Предок всех остальных процессов (ранее назывался <code>init</code>)
crond	Планировщик заданий, который запускает задачи в фоновом режиме
ftpd	Сервер FTP, который обрабатывает клиентские запросы по FTP
httpd	Веб-сервер (Apache), который обрабатывает клиентские запросы по HTTP
sshd	Сервер SSH, который обрабатывает клиентские запросы по SSH

Как правило, названия системных демонов в Linux оканчиваются на `d`. Демонами управляют сценарии оболочки, которые чаще всего хранятся в системном каталоге `/etc/init.d/` (например, в старых версиях Ubuntu) или `/lib/systemd/` (например, в Fedora). Расположение этих файлов зависит от того, как реализована подсистема инициализации, которая управляет всеми службами в Linux.

Обычно эти сценарии оболочки автоматически запускаются при загрузке системы. Но их также можно вызвать с помощью команд управления службами — их обычно запускают уполномоченные пользователи, чтобы регулировать жизненный цикл демонов. Другими словами, пользователь с нужными правами доступа или системный администратор может *остановить* или *запустить* тот или иной демон через интерфейс командной строки. Такие команды немедленно возвращают управление в терминал и выполняют соответствующие операции в фоновом режиме.

Давайте подробнее рассмотрим процесс инициализации.

Процесс инициализации

На протяжении всей этой главы под подсистемой инициализации мы будем подразумевать *обобщенный* механизм инициализации системы и диспетчер служб на платформах Linux. На протяжении многих лет дистрибутивы Linux развивались и использовали различные подсистемы инициализации — например, SysV, upstart, OpenRC, systemd и runit. В сообществе Linux не утихают споры о том, какая из них лучше. Пока что мы будем рассматривать подсистему инициализации просто как системный процесс и разберемся, как он связан с другими процессами.

Подсистема инициализации (которая может называться `init`, `systemd` и т. д.) — это, по сути, системный демон, который запускается одним из первых при загрузке Linux. Процесс этого демона продолжает работать в фоновом режиме, пока система не выключится. Процесс подсистемы инициализации является корневым процессом всех остальных процессов в Linux — то есть родительским процессом или предком более высокого уровня всех процессов в системной иерархии.

В Linux команда `ps tree` отображает полное дерево процессов, которое начинается с процесса подсистемы инициализации. В нашем случае в качестве такого процесса выступает `systemd` (в Ubuntu или Fedora):

```
packt@neptune:~$ ps tree
systemd—ModemManager—2*[{ModemManager}]
      |
      |—agetty
      |—cron
      |—dbus-daemon
      |—multipathd—6*[{multipathd}]
      |—networkd-dispat
      |—polkitd—2*[{polkitd}]
      |—rsyslogd—3*[{rsyslogd}]
      |—snapd—7*[{snapd}]
      |—sshd—sshd—sshd—bash—ps tree
      |—systemd—(sd-pam)
      |—systemd-journal
      |—systemd-logind
      |—systemd-network
      |—systemd-resolve
      |—systemd-timesyn—{systemd-timesyn}
      |—systemd-udev
      |—udisksd—4*[{udis ksd}]
      |—unattended-upgr—{unattended-upgr}
```

Вывод команды `ps tree` иллюстрирует иерархическое дерево процессов, где одни процессы являются родительскими или дочерними по отношению к другим. Давайте рассмотрим иерархические отношения между процессами.

Родительские и дочерние процессы

Когда один процесс порождает другие подчиненные процессы, он называется **родительским** по отношению к ним, а они — **дочерними** по отношению к нему. Дочерние процессы связаны с родительским процессом, который их породил, и обычно завершаются, когда родительский процесс прекращает работу. Впрочем, дочерний процесс может продолжать выполняться и после того, как родительский процесс завершен, если дочерний был настроен (например, с помощью команды `nohup`) так, чтобы игнорировать сигнал `SIGHUP`, который передает родительский процесс при завершении. Дополнительные сведения см. в разделе «Сигналы в Linux».

В Linux каждый процесс, кроме того, который соответствует подсистеме инициализации (во всех ее реализациях), является дочерним по отношению к какому-то другому процессу. Когда завершается дочерний процесс, это само по себе не останавливает родительский. Хорошая практика состоит в том, чтобы завершать родительский процесс, если он больше не нужен, после того как завершился дочерний.

Иногда процессы выполняются без участия пользователя, по определенному расписанию. Такие процессы относятся к пакетной обработке, о которой мы сейчас поговорим.

Пакетные процессы

Пакетный процесс — это, как правило, сценарий или команда, которые автоматически запускаются в определенную дату и время, обычно периодически. Другими словами, пакетная обработка — это фоновый процесс, который порождается **планировщиком задач**. В основном пакетные процессы — это ресурсоемкие задачи, которые обычно назначаются на менее загруженные часы, чтобы не перегружать систему. Чтобы планировать задания, в Linux чаще всего используются `at` и `cron`. С помощью `cron` удобнее управлять сложными задачами, которые запускаются по расписанию, в то время как `at` — это более легкая утилита, которая лучше подходит для разовых заданий. В этой главе мы не будем глубоко знакомиться с этими инструментами. За дополнительной информацией можно обратиться к соответствующим справочным страницам (`man at` и `man cron`).

Заканчивая рассматривать типы процессов, давайте поговорим о процессах-сиротах и процессах-зомби.

Процессы-сироты и процессы-зомби

Когда дочерний процесс завершается, его родительский процесс получает уведомление об этом в виде сигнала `SIGCHLD`. Родительский процесс может продолжить выполнять другие задачи или породить новый дочерний процесс. Однако возможны случаи, когда дочерний процесс продолжает функционировать после того, как его родительский процесс завершился. Такой дочерний процесс называется **процессом-сиротой**, и в Linux его новым родительским процессом автоматически назначается процесс подсистемы инициализации — предок всех процессов.

Процессы-зомби — это ссылки на процессы, которые завершили работу (и передали управление), но все еще остаются в системном списке процессов (который выводит команда `ps`). Основное различие между процессами-зомби и процессами-сиротами состоит в том, что процесс-зомби завершен, а процесс-сирота еще активен.

Различать типы процессов и изучать их поведение помогает то, что значительная часть нужной информации отражена в конструкции самих процессов или их структуре данных. В следующем разделе мы подробнее рассмотрим, из чего состоит процесс. Большую часть этих сведений предоставляет утилита командной строки `ps` — незамысловатое, но очень полезное средство просмотра процессов в системах семейства Linux.

Как устроены процессы

В этом разделе мы рассмотрим некоторые общие свойства процессов Linux с помощью утилит командной строки `ps` и `top`. Надеемся, что практические упражнения помогут вам лучше понять, как устроены процессы, — по крайней мере

с точки зрения администратора Linux. Для начала давайте кратко рассмотрим упомянутые команды. Команда `ps` отображает текущий снимок системных процессов, и ее синтаксис таков:

```
ps [ключи]
```

Без ключей она отображает процессы, которыми владеет текущий сеанс терминала, например:

```
packt@neptune:~$ ps
  PID TTY          TIME CMD
   968 pts/0    00:00:00 bash
  1008 pts/0    00:00:00 ps
```

Давайте рассмотрим поля этого вывода и объясним, что означает каждое поле в контексте нашего текущего процесса, то есть сеанса командной оболочки `bash` в терминале:

- **PID** (сокр. от *process identifier*¹). Это идентификатор, который есть у каждого процесса в Linux и автоматически присваивается ядром при создании процесса. Значение **PID** — целое число, которое уникально для каждого процесса. В нашем случае мы рассматриваем процесс `bash` (текущую оболочку) с **PID** 968.
- **TTY**. Как упоминалось в разделе «Как подключиться к командной оболочке» (глава 2), аббревиатурой **TTY** обозначается управляющий терминал, то есть устройство, с помощью которого пользователь взаимодействует с системой. В контексте процессов Linux поле **TTY** обозначает тип терминала, с которым связан процесс. В нашем примере процессу `bash`, который представляет сеанс командной оболочки, соответствует терминал `pts/0`. Здесь `pts` означает, что ввод и вывод процесса контролирует псевдотерминал, а `/0` — порядковый номер соответствующего терминального сеанса. Например, если открыть еще один сеанс SSH, то его терминал будет обозначаться как `pts/1`, и т. д.
- **TIME**. Это совокупные затраты процессорного времени на процесс (в формате `[ДД-]ЧЧ:ММ:СС`). Почему у процесса `bash` в нашем примере это значение равно нулю (`00:00:00`)? Бывает так, что в сеансе терминала выполняется много команд, а загрузка процессора все равно остается нулевой. Это происходит потому, что поле **TIME** показывает, сколько времени тратится на выполнение каждой отдельной команды, а не родительского сеанса терминала в целом. Если команды завершаются за доли секунды, они вряд ли приведут к тому, что в поле **TIME** накопится ощутимое количество процессорного времени.
- **CMD** (сокр. от *command*). Это поле содержит имя или полный путь к команде, которая создала процесс. Для стандартных системных команд (например, `bash`) поле **CMD** отображает имя команды вместе с аргументами.

¹ Идентификатор процесса. — *Примеч. пер.*

Поля, которые мы рассмотрели до сих пор, дают относительно простое представление о процессах Linux. Но бывают ситуации, когда нам нужно больше информации. Например, команда `ps -l` предоставляет дополнительные сведения о процессах, которые запущены в текущем сеансе терминала:

```
F S    UID      PID      PPID  C  PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S   1000      968      967  0   80   0 -  2213 do_wai pts/0      00:00:00 bash
0 R   1000     1058      968  0   80   0 -  2517 -      pts/0      00:00:00 ps
```

Команда `ps` с ключом `-l` выводит сведения о процессах в расширенном формате. Вот наиболее важные поля этого формата:

F	Флаги процесса (например: 0 — нет флагов, 1 — процесс порожден системным вызовом <code>fork</code> , но еще не вызвал <code>exec</code> , 4 — процесс запущен с полномочиями суперпользователя)
S	Код состояния процесса (например: R — процесс выполняется, S — режим прерываемого ожидания)
UID	Идентификатор пользователя — владельца процесса
PID	Идентификатор процесса
PPID	Идентификатор родительского процесса
PRI	Приоритет процесса (чем больше число, тем ниже приоритет)
SZ	Использование виртуальной памяти

У процессов есть еще немало других атрибутов, но в этой книге мы не будем в них углубляться. За дополнительной информацией обратитесь к справочной странице команды `ps` (`man ps`).

До сих пор в примерах использования команды `ps` мы рассматривали только те процессы, которыми владеет текущий сеанс терминала. Мы считаем, что такой подход позволяет объяснить основные свойства процессов без лишних сложностей.

Есть и еще одна команда — `top`, с помощью которой можно просматривать сведения обо всех запущенных процессах в системе в режиме реального времени. Ее синтаксис таков:

`top [ключи]`

В выводе команды `top` отображаются многие поля, которые уже встречались нам в контексте `ps`, хотя некоторые из них обозначаются по-другому. Давайте рассмотрим, что означает каждое поле. Команда `top` без ключей выводит сведения о запущенных процессах в реальном времени, например:

```
top - 12:02:40 up 32 min,  1 user,  load average: 0.26, 0.18, 0.12
Tasks: 113 total,  1 running, 112 sleeping,  0 stopped,  0 zombie
```

```
%Cpu(s):  0.0 us,  6.2 sy,  0.0 ni, 93.8 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem : 1976.0 total, 1424.6 free,  222.1 used,  329.3 buff/cache
MiB Swap: 1840.0 total, 1840.0 free,    0.0 used. 1604.8 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
220	root	20	0	166492	11736	8288	S	0.0	0.6	0:02.13	systemd
1	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kthreadd
2	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	rcu_gp
11	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	rcu_par+
53	root	0	-20	0	0	0	I	0.0	0.2	0:00.00	slub_flt
94	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	netns

Рассмотрим некоторые поля этого вывода:

USER	Имя пользователя, который владеет процессом
PR	Приоритет процесса
NI	«Добропорядочность» (niceness) процесса — своего рода механизм адаптивной приоритизации
VIRT	Объем виртуальной памяти, которую использует процесс (в килобайтах)
RES	Объем резидентной памяти (без подкачки), которую использует процесс (в килобайтах)
SHR	Объем памяти, который процесс использует совместно с другими процессами (в килобайтах)
S	Состояние процесса (например: R — процесс выполняется, S — режим прерываемого ожидания, I — процесс простаивает)
%CPU	Потребление ресурсов процессора (в процентах)
%MEM	Потребление резидентной памяти (в процентах)
COMMAND	Имя соответствующей команды

Каждое из этих полей (а также многие другие) подробно описано на справочной странице команды `top` (`man top`).

В повседневных задачах администрирования Linux часто приходится запрашивать информацию о процессах, обращаясь к ранее перечисленным полям. В разделе «Как работать с процессами» мы рассмотрим наиболее характерные примеры использования команд `ps` и `top`, а также другие профессиональные приемы.

Один из важнейших атрибутов процесса — его **состояние** (или **статус**). Нас часто интересует состояние процесса в тот или иной момент времени, а также переходы из одного состояния в другое. Команды `ps` и `top` предоставляют сведения о состоянии процессов с помощью поля `S`. Давайте рассмотрим состояния процессов подробнее.

Состояния процесса

В течение своего жизненного цикла процесс может переходить из одного состояния в другое в зависимости от обстоятельств. Его текущее состояние отображается в поле *S* команд *ps* и *top*. Вот список возможных состояний процесса:

D	Непрерываемое ожидание
I	Простаивает
R	Выполняется
S	Прерываемое ожидание
T	Остановлен сигналом управления заданием
t	Остановлен отладчиком во время трассировки
Z	Процесс-зомби

На высоком уровне любое из этих состояний можно отождествить с одним из таких положений:

- **Процесс функционирует.** Процесс выполняется (R) или простаивает (I). В Linux простаивающий процесс — это определенная задача, которая назначается каждому процессору в системе и планируется к выполнению только тогда, когда на этом процессоре не запущены никакие другие процессы. Время, которое тратится на простаивающие задачи, учитывается в атрибуте `id` строки `%Cpu(s)`, которую выводит команда *top*.
- **Режим ожидания.** Процесс ожидает до тех пор, пока не произойдет определенное событие или пока не станет доступен определенный ресурс. Ожидание бывает прерываемым (S) и непрерываемым (D). Процесс можно вывести из прерываемого ожидания, передав ему определенные сигналы, после чего он будет выполняться дальше. Наоборот, непрерываемое ожидание — это состояние, когда процесс заблокирован в системном вызове (возможно, в ожидании каких-то аппаратных условий) и его невозможно прервать.
- **Процесс остановлен.** Процесс перестал выполняться, обычно в результате того или иного сигнала — сигнала управления заданием (T) или сигнала отладки (t).
- **Зомби.** Процесс не функционирует или «умер» (состояние Z): он завершен, но родительский процесс его не утилизировал. Процесс-зомби — это, по сути, «мертвая» ссылка на уже завершенный процесс в системной таблице процессов. Подробнее об этом говорилось в подразделе «Процессы-сироты и процессы-зомби».

Заканчивая разговор про состояния процессов, давайте рассмотрим жизненный цикл процесса в Linux. Обычно процесс начинается с состояния выполнения (R)

и завершается, когда родительский процесс утилизирует его из состояния зомби (Z). Следующая диаграмма дает общее представление о том, в каких состояниях может находиться процесс и как он переходит из одного состояния в другое:

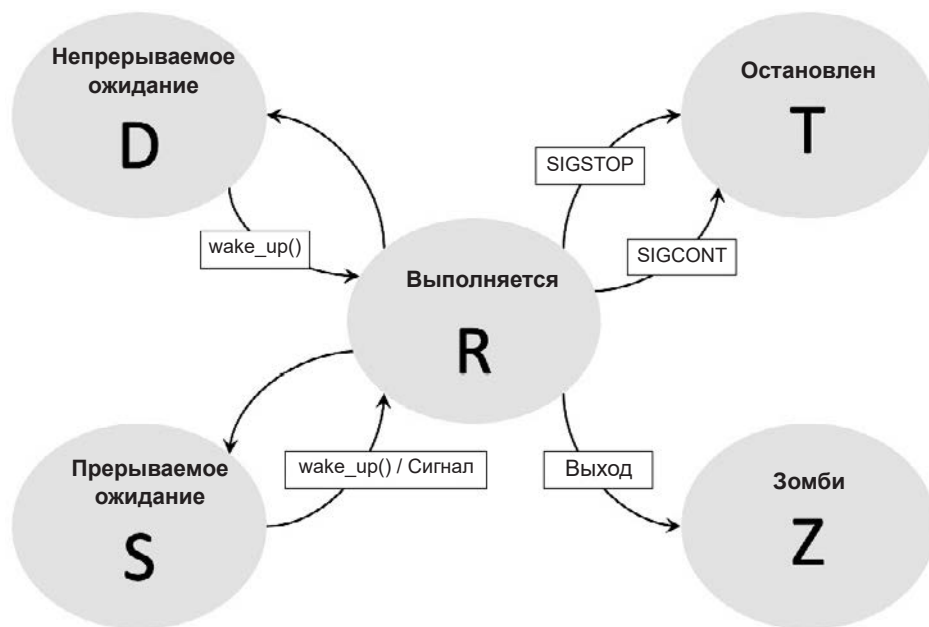


Рис. 5.1. Жизненный цикл процесса в Linux

Теперь, когда мы представили процессы и в общих чертах рассказали о том, каких типов они бывают и как они устроены, можно попробовать взаимодействовать с ними. В следующих разделах мы рассмотрим некоторые стандартные утилиты командной строки, которые позволяют работать с процессами и демонами. Большинство этих утилит используют в качестве входных и выходных данных поля, которые мы рассматривали в разделе «Как устроены процессы». Итак, давайте поработаем с процессами.

Как работать с процессами

Этот раздел научит вас на практике управлять процессами с помощью удобных утилит командной строки, которые используются в повседневных задачах администрирования Linux. Некоторые из этих утилит (например, `ps` и `top`) уже упоминались в предыдущих разделах, когда мы рассказывали о внутреннем устройстве процессов. Здесь мы обобщим большую часть предыдущего материала и рассмотрим практические примеры.

Начнем с команды `ps`, которая служит средством просмотра процессов в Linux.

Как просматривать сведения о процессах с помощью `ps`

Мы представили команду `ps` и описали ее синтаксис в разделе «Как устроены процессы». Следующая команда выводит выборку из множества процессов, запущенных в системе:

```
ps -e | head
```

Ключ `-e` (или идентичный ему `-A`) выбирает *все* процессы в системе. Вывод перенаправляется в команду `head`, которая отображает только несколько первых строк (по умолчанию 10):

```
packt@neptune:~$ ps -e | head
  PID TTY          TIME CMD
    1 ?            00:00:02 systemd
    2 ?            00:00:00 kthreadd
    3 ?            00:00:00 rcu_gp
    4 ?            00:00:00 rcu_par_gp
    5 ?            00:00:00 slub_flushwq
    6 ?            00:00:00 netns
    7 ?            00:00:01 kworker/0:0-events
    8 ?            00:00:00 kworker/0:0H-events_highpri
   10 ?            00:00:00 mm_percpu_wq
```

Такие сведения иногда оказываются не особенно полезными. Возможно, нам понадобится больше информации о каждом процессе, чем значения полей `PID` или `CMD`. (Мы рассматривали их в разделе «Как устроены процессы».)

Следующая команда более детально перечисляет процессы текущего пользователя:

```
ps -fu $(whoami)
```

Ключ `-f` выводит данные в расширенном формате, отображая подробные сведения о каждом процессе. Ключ `-U $(whoami)` обозначает, что нас интересуют процессы, которыми владеет текущий пользователь (в нашем случае `packt`). Другими словами, мы хотим получить список всех своих процессов:

```
packt@neptune:~$ ps -fu $(whoami)
  UID      PID  PPID  C  STIME TTY          TIME CMD
packt      864     1   0  11:30 ?          00:00:00 /lib/systemd/systemd --user
packt      865    864   0  11:30 ?          00:00:00 (sd-pam)
packt      967    861   0  11:30 ?          00:00:00 sshd: packt@pts/0
packt      968    967   0  11:30 pts/0      00:00:00 -bash
packt     1079    968   0  12:31 pts/0      00:00:00 ps -fu packt
```

Бывают ситуации, когда нужно найти определенный процесс, чтобы пронаблюдать за ним или воздействовать на него. Давайте вспомним недавний пример,

где мы демонстрировали долговременный процесс, и обернем похожую команду в простой сценарий. Команда представляет собой элементарный цикл `while`, который выполняется бесконечно:

```
while true; do x=1; done
```

Чтобы создать сценарий, откройте свой любимый текстовый редактор (например, `nano`) и создайте в нем файл (например, `test.sh`):

```
packt@neptune:~$ nano test.sh
```

Наберите такой код и сохраните файл:

```
#!/bin/bash
while true; do x=1; done
```

Этот тестовый сценарий можно сделать исполняемым и запустить как фоновый процесс — обратите внимание, что для этого служит амперсанд (`&`) в конце команды:

```
chmod +x test.sh
./test.sh &
[1] 1094
```

Фоновому процессу, в котором выполняется наш сценарий, соответствует PID 1094. Предположим, что мы хотим получить сведения о процессе по имени файла (`test.sh`). Для этого можно запустить конвейер из команд `ps` и `grep`:

```
packt@neptune:~$ ps -ef | grep test.sh
packt      1094      968  99 12:41 pts/0    00:02:28 /bin/bash ./test.sh
packt      1097      968   0 12:44 pts/0    00:00:00 grep --color=auto test.sh
```

Этот вывод показывает, что нашему процессу соответствует PID 1094, а поле CMD содержит `/bin/bash ./test.sh` — это полный вызов команды, которая запустила наш сценарий, включая аргументы командной строки.

Обратите внимание, что первая строка сценария `test.sh` выглядит как `#!/bin/bash` — это директива, которая заставляет операционную систему вызвать `bash`, чтобы выполнить сценарий. Эта строка также известна как шебанг-строка, и она должна быть первой в сценарии командной оболочки. Что касается поля CMD, то в нашем случае его значение начинается с команды `/bin/bash` (как указано в шебанг-строке), а аргумент этой команды — имя сценария `test.sh`. Другими словами, программа `bash` выполняет сценарий `test.sh`.

Предыдущая команда `ps` вывела также сведения о процессе `ps | grep`, который не имеет отношения к текущей задаче. Уточним эту же команду таким образом:

```
packt@neptune:~$ ps -ef | grep test.sh | grep -v grep
packt      1094      968  99 12:41 pts/0    00:04:57 /bin/bash ./test.sh
```

Здесь команда `grep -v grep` в конце конвейера отфильтровывает лишний вызов `grep` из вывода команды `ps`.

Чтобы найти процесс по его PID, можно вызвать команду `ps` с параметром `-p` (`--pid`). Например, такая команда выводит подробную информацию о нашем процессе с PID 1094 (сценарий `test.sh`, который мы недавно запустили):

```
packt@neptune:~$ ps -fp 1094
UID          PID    PPID  C  STIME TTY          TIME CMD
packt        1094      968  99  12:41 pts/0      00:06:50 /bin/bash ./test.sh
```

Ключ `-f` выводит расширенные сведения о процессе.

Команду `ps` можно использовать самыми разнообразными способами, но в этой книге мы ограничились наиболее типичными примерами, которые послужат вам ориентиром в самостоятельных изысканиях. За более подробной информацией обращайтесь к справочной странице команды `ps` (`man ps`).

Как просматривать дерево процессов с помощью команды `pstree`

Команда `pstree` отображает выполняющиеся процессы в иерархическом, древовидном формате — это своего рода визуальное дополнение команды `ps`. Дерево `pstree` начинается либо с процесса подсистемы инициализации (например, `systemd`), либо с того процесса, PID которого передан команде в качестве аргумента. Ее синтаксис таков:

```
pstree [ключи] [PID] [имя_пользователя]
```

Следующая команда отображает дерево процессов текущего сеанса терминала:

```
packt@neptune:~$ pstree $(echo $$)
bash─┬─pstree
      └─test.sh
```

В предыдущей команде `echo $$` возвращает PID текущего сеанса терминала. `$$` — это встроенная переменная Bash, которая содержит значение PID текущей оболочки. Здесь оно передается в качестве аргумента команде `pstree`. Чтобы просмотреть PID процессов, можно запустить `pstree` с ключом `-p` (`--show-pids`):

```
packt@neptune:~$ pstree $(echo $$)
bash(968)─┬─pstree(1117)
          └─test.sh(1094)
```

Следующая команда выводит процессы, которыми владеет текущий пользователь:

```
packt@neptune:~$ pstree $(whoami)
sshd─┬─bash─┬─pstree
      │      └─test.sh
```

Чтобы получить дополнительную информацию о команде `pstree`, обратитесь к соответствующей справочной странице (`man pstree`).

Как просматривать сводку процессов с помощью команды `top`

Когда нужно наблюдать за процессами в режиме реального времени, администраторы Linux чрезвычайно часто используют утилиту `top`. Ее синтаксис таков:

```
top [ключи]
```

Команда `top` без ключей отображает все процессы, которые выполняются в системе, и в реальном времени обновляет сведения о них (потребление памяти, процессора и т. д.):

```
top
```

Чтобы выйти из `top`, нажмите `q`. По умолчанию процессы сортируются по потреблению ресурсов процессора (этому показателю соответствует поле `%CPU`).

Если нужно сортировать процессы по другому полю, во время работы `top` нажмите `Shift + F`, чтобы войти в интерактивный режим. В этом режиме с помощью клавиш со стрелками можно выбрать нужное поле для сортировки (например, `%MEM`), а затем нажать `s`, чтобы назначить его, и `q`, чтобы выйти из интерактивного режима.

Вместо того чтобы настраивать сортировку в интерактивном режиме, можно задать поле для сортировки с помощью ключа `-o`. Например, такая команда выводит список из 10 процессов, которые активнее всего потребляют ресурсы процессора:

```
top -b -o %CPU | head -n 17
```

А такая команда выводит список из 10 процессов, отсортированных по потреблению процессора и памяти:

```
top -b -o +%MEM | head -n 17
```

Ключ `-b` вызывает пакетный режим работы (вместо режима реального времени по умолчанию), в котором команда однократно выводит текущий список процессов и возвращает управление. Ключ `-o +%MEM` задает добавочное (+) поле сортировки (`%MEM`) в дополнение к стандартному полю `%CPU`. Ключ `head -n 17` выбирает первые 17 строк вывода (из них 7 строк занимает заголовок вывода `top`):

```
packt@neptune:~$ top -b -o +%MEM | head -n 17
top - 12:59:24 up 1:29, 1 user, load average: 1.00, 0.99, 0.71
Tasks: 115 total, 2 running, 113 sleeping, 0 stopped, 0 zombie
%Cpu(s):100.0 us, 0.0 sy, 0.0 ni, 0.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
MiB Mem : 1976.0 total, 1417.6 free, 211.7 used, 346.6 buff/cache
MiB Swap: 1840.0 total, 1840.0 free, 0.0 used. 1612.4 avail Mem
```

PID	USER	PR	NI	VRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1013	root	20	0	500400	64388	22584	S	0.0	3.2	0:00.74	fwupd
665	root	20	0	727508	40732	20716	S	0.0	2.0	0:01.18	snappd
421	root	rt	0	289312	27360	9072	S	0.0	1.4	0:00.49	multitapd
720	root	20	0	109748	21292	13136	S	0.0	1.1	0:00.05	unattended
661	root	20	0	32652	18932	10284	S	0.0	0.9	0:00.04	networkd
381	root	19	-1	47856	15664	14572	S	0.0	0.8	0:00.12	systemd
669	root	20	0	392564	12880	10792	S	0.0	0.6	0:00.06	udisksd
641	systemd	20	0	25260	12452	8516	S	0.0	0.6	0:00.04	systemd
705	root	20	0	317948	11964	10060	S	0.0	0.6	0:00.03	ModemManager
1	root	20	0	166492	11736	8288	S	0.0	0.6	0:02.16	systemd

Следующая команда перечисляет пять процессов, которыми владеет текущий пользователь (`packt`) и которые активнее всего потребляют процессор:

```
top -u $(whoami) -b -o %CPU | head -n 12
```

Ключ `-u $(whoami)` заставляет команду `top` выводить только процессы текущего пользователя.

С помощью `top` также можно отслеживать конкретные процессы, если передавать PID через ключ `-p`. Например, такая команда отслеживает наш тестовый процесс (с PID 1094):

```
packt@neptune:~$ top -p 1094
top - 13:02:58 up 1:32, 1 user, load average: 1.00, 1.00, 0.78
Tasks: 1 total, 1 running, 0 sleeping, 0 stopped, 0 zombie
%Cpu(s):100.0 us, 0.0 sy, 0.0 ni, 0.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
MiB Mem : 1976.0 total, 1417.6 free, 211.7 used, 346.7 buff/cache
MiB Swap: 1840.0 total, 1840.0 free, 0.0 used. 1612.4 avail Mem
```

PID	USER	PR	NI	VRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1094	packt	20	0	7368	1468	1312	R	99.3	0.1	21:13.64	test.sh

Если `top` работает в режиме реального времени, можно принудительно завершить любой процесс, нажав `K`. При этом система предложит ввести PID целевого процесса:

```
top - 13:04:37 up 1:34, 1 user, load average: 1.00, 1.00, 0.81
Tasks: 1 total, 1 running, 0 sleeping, 0 stopped, 0 zombie
%Cpu(s):100.0 us, 0.0 sy, 0.0 ni, 0.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
MiB Mem : 1976.0 total, 1417.6 free, 211.7 used, 346.7 buff/cache
MiB Swap: 1840.0 total, 1840.0 free, 0.0 used. 1612.4 avail Mem
PID to signal/kill [default pid = 1094] _
PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND
1094 packt 20 0 7368 1468 1312 R 99.9 0.1 22:52.66 test.sh
```

У утилиты `top` весьма широкая область применения. Мы надеемся, что примеры из этого раздела вдохновили вас на то, чтобы изучать этот инструмент подробнее и эффективно использовать его для своих задач. За дополнительной информацией обращайтесь к соответствующей справочной странице (`man top`).

Как принудительно завершать процессы с помощью команд `kill` и `killall`

Команду `kill` чаще всего применяют для того, чтобы завершить процесс. Ее синтаксис таков:

```
kill [ключи] [ -s сигнал | -сигнал ] PID [...]
```

Команда `kill` передает процессу *сигнал* — например, чтобы попытаться остановить процесс¹. Если сигнал не указан, передается `SIGTERM` (15). Чтобы задать сигнал, можно указать либо его имя без префикса `SIG` (например, `KILL` для сигнала `SIGKILL`), либо соответствующее числовое значение (например, 9 для `SIGKILL`).

Команда `kill` с ключом `-l` или `-L` выводит полный список сигналов, которые доступны в Linux:

```
packt@neptune:~$ kill -L
 1) SIGHUP      2) SIGINT      3) SIGQUIT     4) SIGILL      5) SIGTRAP
 6) SIGABRT    7) SIGBUS     8) SIGFPE     9) SIGKILL    10) SIGUSR1
11) SIGSEGV   12) SIGUSR2   13) SIGPIPE   14) SIGALRM   15) SIGTERM
16) SIGSTKFLT 17) SIGCHLD  18) SIGCONT   19) SIGSTOP   20) SIGTSTP
21) SIGTTIN   22) SIGTTOU  23) SIGURG    24) SIGXCPU   25) SIGXFSZ
26) SIGVTALRM 27) SIGPROF  28) SIGWINCH  29) SIGIO     30) SIGPWR
31) SIGSYS    34) SIGRTMIN 35) SIGRTMIN+1 36) SIGRTMIN+2 37) SIGRTMIN+3
38) SIGRTMIN+4 39) SIGRTMIN+5 40) SIGRTMIN+6 41) SIGRTMIN+7 42) SIGRTMIN+8
43) SIGRTMIN+9 44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9 56) SIGRTMAX-8 57) SIGRTMAX-7
58) SIGRTMAX-6 59) SIGRTMAX-5 60) SIGRTMAX-4 61) SIGRTMAX-3 62) SIGRTMAX-2
63) SIGRTMAX-1 64) SIGRTMAX
```

Предыдущий вывод показывает, какое числовое значение соответствует каждому сигналу, — например, для `SIGKILL` оно равно 9. Следующая команда приведет к тому, что наш тестовый процесс с PID 1094 завершится:

```
kill -9 1094
```

К тому же результату приведет и такая команда:

```
kill -KILL 1094
```

¹ Команда `kill` впервые появилась в 3-м издании ОС Unix (февраль 1973) и служила только для того, чтобы принудительно завершить процесс, — отсюда название `kill` (англ. *убить*). Однако уже в 4-м издании Unix (ноябрь 1973) команда `kill` позволяла передать процессу произвольный сигнал, и с тех пор в системах семейства Unix и Linux появились десятки сигналов, большинство из которых не относятся к завершению процесса. Название команды `kill` сохраняется по историческим соображениям. — *Примеч. ред.*

Иногда бывает нужно завершить сразу несколько процессов. Здесь на помощь приходит команда `killall`. Ее синтаксис таков:

```
killall [ключи] [ -s сигнал | -сигнал ] команда...
```

`killall` передает сигнал всем процессам, которые выполняют любую из указанных команд. По умолчанию передается сигнал `SIGTERM` (15). Как и в случае с `kill`, сигнал можно задать либо в виде имени без префикса, либо в виде числового значения (например, `TERM` или 15 для `SIGTERM`).

Например, следующая команда завершает все процессы, которые выполняют сценарий `test.sh`:

```
packt@neptune:~$ killall -e -TERM test.sh
```

```
packt@neptune:~$ ps
  PID TTY          TIME CMD
   968 pts/0    00:00:00 bash
  1173 pts/0    00:00:00 ps
[1]+  Terminated                ./test.sh
```

После того как процесс завершен, соответствующая ссылка обычно удаляется из системной таблицы процессов, и завершенный процесс больше не отображается в выводе `ps`, `top` или аналогичных команд.

Чтобы получить дополнительную информацию о командах `kill` и `killall`, обратитесь к соответствующим справочным страницам (`man kill` и `man killall`).

Использование команд `pgrep` и `pkill`

Команды `pgrep` и `pkill` позволяют искать запущенные процессы по шаблону и завершать их. Синтаксис обеих команд одинаковый:

```
pgrep [ключи] шаблон
pkill [ключи] шаблон
```

`pgrep` выбирает из всех текущих процессов те, которые соответствуют шаблону или критериям отбора, а `pkill` передает подходящим процессам сигналы (например, принудительно завершает их).

Например, команда `pgrep -f test.sh` ищет наш тестовый процесс (`test.sh`) и выводит его `PID`, если процесс найден. Прежде чем вводить эту команду, запустите процесс снова, потому что в предыдущем разделе мы его завершили. Обратите внимание, что теперь у процесса другой `PID`:

```
packt@neptune:~$ ./test.sh &
[1] 1178

packt@neptune:~$ pgrep -f test.sh
1178
```

Ключ `-f (--full)` ищет процессы по полному соответствию команды, которая их запустила, а не просто по имени процесса. Чтобы получить более подробные сведения о процессе, `pgrep` можно использовать вместе с `ps`, например:

```
packt@neptune:~$ pgrep -f test.sh | xargs ps -fp
UID      PID     PPID  C  STIME TTY          TIME CMD
packt    1178     968  99  13:15 pts/0      00:02:07 /bin/bash ./test.sh
```

В предыдущем примере мы передали вывод команды `pgrep` (с PID 1178) команде `ps`, которая запускается с ключами `-f` (расширенный формат) и `-p` (PID). Ключ `-p` принимает значение PID, которое передано по конвейеру.

Команда `xargs` принимает то, что выводит `pgrep`, и преобразует эти данные в аргумент для команды `ps`. Таким образом, конвейер `pgrep | ps` автоматически преобразует вывод первой команды в аргумент второй команды.

Чтобы завершить процесс `test.sh`, можно просто вызвать команду `kill`:

```
kill -f test.sh
```

Предыдущая команда завершает процесс, которому соответствует полная команда, указанная после ключа `-f`, и не выводит никаких данных. Чтобы получить обратную связь от `kill`, эту команду нужно запустить с ключом `-e (--echo)`:

```
packt@neptune:~$ kill -ef test.sh
test.sh killed (pid 1178)1
[1]+  Terminated                  ./test.sh
```

Чтобы получить дополнительную информацию о командах `pgrep` и `kill`, обратитесь к соответствующим справочным страницам (`man pgrep` и `man kill`).

В этом разделе мы рассмотрели популярные утилиты командной строки, с помощью которых обычно управляют процессами в повседневных задачах администрирования Linux. Помните, что одни и те же задачи в Linux чаще всего можно выполнить многими способами. Мы надеемся, что примеры из этого раздела помогут вам придумывать собственные эффективные методы и приемы работы с процессами.

Далее мы рассмотрим некоторые особенности взаимодействия с демонами.

Как работать с демонами

Как отмечалось во вступительных разделах, демоны — это особый вид фоновых процессов, а значит, к ним применимо подавляющее большинство методов и приемов работы с процессами. Однако есть особые команды, которые при-

¹ Процесс `test.sh` с PID 1178 принудительно завершен. — *Примеч. пер.*

меняются исключительно к демонам, когда нужно управлять жизненным циклом соответствующих процессов.

Как отмечалось в разделе «Что такое демоны», процессами демонов управляют сценарии оболочки, которые обычно хранятся в системных каталогах `/etc/init.d/` или `/lib/systemd/`, в зависимости от дистрибутива. В старых системах Linux (например, RHEL 6) файлы сценариев демонов хранились в `/etc/init.d/`, а начиная с RHEL 7 и Ubuntu 18.04, они обычно расположены в `/lib/systemd/`. Просмотрите содержимое этих двух каталогов с помощью `ls`.

То, где находятся файлы демонов и соответствующие утилиты командной строки, во многом зависит от подсистемы инициализации и системы управления службами. В разделе «Процесс инициализации» (глава 1) мы вкратце упомянули о различных системах инициализации в дистрибутивах Linux. Чтобы продемонстрировать, как работают команды управления демонами, мы рассмотрим систему инициализации `systemd`, которая широко используется на различных платформах Linux.

Как работать с демонами `systemd`

Основная задача подсистемы инициализации — инициировать запуск различных процессов и координировать их зависимости, когда загружается ядро Linux. Большинство этих процессов выполняется в **пользовательском пространстве**. Подсистема инициализации также управляет службами и демонами во время работы системы.

За последние несколько лет большинство платформ Linux перешли на подсистему инициализации `systemd`, поэтому крайне важно познакомиться с ней и соответствующими инструментами командной строки. Соответственно, в текущем разделе рассматривается прежде всего `systemctl` — главная утилита командной строки, которая позволяет управлять демонами `systemd`.

Синтаксис `systemctl` таков:

```
systemctl [ключи] [команда] [модули...]
```

Команда `systemctl` работает с *модулями* — системными ресурсами, которыми управляет `systemd`. Модули бывают разных типов — например, `service` (служба), `mount` (точка монтирования файловой системы), `socket` и т. д. Каждому типу соответствуют конфигурационные файлы с одноименным суффиксом: например, `httpd.service` — это файл модуля веб-службы (демона) Apache. Все типы модулей `systemd` перечислены и подробно описаны на справочной странице `systemd.unit` (`man systemd.unit`).

Следующая команда служит для того, чтобы демон (например, веб-сервер `httpd`) запускался при загрузке системы:

```
sudo systemctl enable httpd
```

Чтобы запускать команду `systemctl`, обычно нужны права суперпользователя. Обратите внимание, что `systemctl` не требует суффикса `.service`, когда мы обращаемся к модулям. Предыдущую команду можно было вызвать и так:

```
sudo systemctl enable httpd.service
```

Наоборот, чтобы служба `httpd` перестала запускаться при загрузке, можно запустить такую команду:

```
sudo systemctl disable httpd
```

А такая команда позволяет запросить состояние службы `httpd`:

```
sudo systemctl status httpd
```

Состояние службы `httpd` также можно проверить с помощью такой команды:

```
sudo systemctl is-active httpd
```

Команды `start` и `stop` соответственно однократно запускают и останавливают службу `httpd`:

```
sudo systemctl start httpd  
sudo systemctl stop httpd
```

Дополнительные сведения о `systemctl` вы найдете на соответствующей справочной странице (`man systemctl`). А чтобы подробнее узнать, как устроена подсистема инициализации `systemd`, обратитесь к ее справке (`man systemd`).

Системному администратору Linux постоянно приходится иметь дело с процессами и демонами в повседневной работе, поэтому ему необходимо владеть соответствующими утилитами командной строки. Однако не менее важно понимать, как те или иные выполняющиеся процессы или демоны связаны с другими процессами или демонами, которые запущены как локально, так и на удаленных узлах. То, как процессы взаимодействуют друг с другом, может поначалу показаться «черной магией». Мы разоблачим эту «магию» в следующем разделе, где поговорим о межпроцессном взаимодействии.

Межпроцессное взаимодействие

Межпроцессное взаимодействие (IPC, inter-process communication) позволяет процессам обмениваться данными с помощью общего механизма или интерфейса. В этом разделе мы в общих чертах рассмотрим, как происходит эта коммуникация. Некоторые вопросы межпроцессного взаимодействия подробнее рассматриваются в главе 8 «Сценарии командной оболочки в Linux».

Процессы в Linux чаще всего обмениваются данными и синхронизируют свои действия через такие интерфейсы:

- **Общее хранилище (файлы).** Самое простое общее хранилище IPC может быть обычным файлом, который находится на диске. Процесс-отправитель записывает данные в файл, а процесс-получатель читает их оттуда. В этом случае очевидная трудность состоит в том, как обеспечить целостность чтения и записи, чтобы соответствующие низкоуровневые операции не оказывались в ситуации гонки. Если не блокировать файл во время записи, то операции ввода-вывода могут перекрываться другими операциями чтения или записи. Чтобы не усложнять материал, в наших учебных примерах мы не будем решать проблему гонки, но в реальной работе о ней не стоит забывать.
- **Общая память.** У каждого процесса в Linux обычно есть отдельное адресное пространство. Процесс может получить доступ к данным в памяти другого процесса, только если у них есть общий сегмент памяти, в котором хранятся эти данные. Linux предоставляет несколько API, с помощью которых можно программно определять общую память процессов и управлять ею: например, старый API System V и более современный API POSIX. Они оба написаны на языке C, хотя реализация модели отправителя и получателя выходит за рамки этой книги. Впрочем, мы можем в некотором роде поработать с общей памятью, если будем обращаться к файлам в каталоге `/dev/shm`, содержимое которого на самом деле хранится в оперативной памяти системы по принципу RAM-диска.

Если в качестве общей памяти используется `/dev/shm`, то можно опираться на ту же модель «отправитель — получатель», о которой шла речь в предыдущем пункте («Общее хранилище»), и просто размещать общий файл в этом каталоге — например, под именем `/dev/shm/storage`.

Модели IPC с общей памятью и общим хранилищем могут оказаться неэффективными, когда нужно работать с большими объемами данных, особенно если они представлены в виде потока. Здесь на помощь приходят каналные механизмы IPC, которые могут реализовываться как каналы, очереди сообщений или сокет.

- **Именованные и неименованные каналы.**
 - **Неименованные каналы** передают выходные данные одного процесса на вход другого. В рамках нашей модели «отправитель — получатель» неименованный канал как механизм коммуникации между двумя процессами можно проиллюстрировать так:

`отправитель.sh | получатель.sh`

Ключевой компонент этого кода — вертикальная черта (`|`). Процесс, которому соответствует команда слева от черты, вырабатывает данные, которые поступают на вход второго процесса.
 - **Именованные каналы**, также известные как **FIFO**, на первый взгляд похожи на неименованные, но представляют другую семантику. Неименованный канал сохраняется только до тех пор, пока работает связанный

с ним процесс. Наоборот, у именованного канала есть собственное хранилище данных, и по умолчанию он существует до тех пор, пока система функционирует, — независимо от того, активны ли процессы, которые с ним связаны. Обычно именованный канал ведет себя как файл, и его можно удалить, если он больше не нужен.

- **Очереди сообщений.** Очередь сообщений — это механизм асинхронной коммуникации, который обычно используется в распределенных системах. Сообщения фиксируются и хранятся в очереди до тех пор, пока не будут обработаны и в конечном итоге удалены. Отправитель создает (публикует) сообщение, а затем его однократно обрабатывает получатель (обычно единственный). На самом высоком уровне сообщению соответствует серия, полезная нагрузка и тип. Очереди сообщений могут регулировать то, в каком порядке передаются сообщения (например, в зависимости от приоритета или типа):



Рис. 5.2. Очередь сообщений (упрощенный вид)

В этой главе мы не станем подробно анализировать очереди сообщений или моделировать их реализацию. Существует много реализаций очередей с открытым исходным кодом, которые доступны для большинства платформ Linux: RabbitMQ, ActiveMQ, ZeroMQ, MQTT и т. д.

Механизмы межпроцессной коммуникации, которые основаны на очередях сообщений и каналах, являются однонаправленными: один процесс передает данные, а другой их принимает. Существуют двунаправленные реализации именованных каналов, но с ними связаны технические сложности, которые затрудняют обмен данными. Для двунаправленной коммуникации можно использовать механизм IPC на основе сокетов (подробнее см. в главе 8 «Сценарии командной оболочки в Linux»).

- **Сокеты.** Для межпроцессной коммуникации используются сокеты двух типов:
 - **Сокеты IPC** также известны как доменные сокеты Unix: они используют локальный файл в качестве адреса сокета и позволяют процессам на одном узле обмениваться данными в обоих направлениях.

- **Сетевые сокеты** расширяют возможности межпроцессного взаимодействия за пределы локального компьютера через сетевые соединения TCP или UDP.

Не считая очевидных различий в реализации, сокеты IPC и сетевые сокеты ведут себя одинаково. Сокеты обоих типов настроены как потоки, поддерживают двунаправленную связь и эмулируют модель «клиент — сервер». Соединение через сокет активно до тех пор, пока его не закроет одна из сторон.

- **Сигналы.** В Linux сигнал — это односторонний асинхронный механизм уведомлений, с помощью которых различные компоненты системы реагируют на те или иные события. Сигналы могут передаваться в любом из направлений:
 - от ядра Linux к произвольному процессу;
 - от процесса к процессу;
 - от процесса к самому себе.

В начале этого раздела мы упомянули, что сигналы — это один из механизмов IPC. Они действительно представляют собой своего рода ограниченную форму IPC в том смысле, что с помощью сигналов процессы могут координировать свою активность между собой. Но сигналы не несут никакой полезной нагрузки в виде данных, а просто уведомляют процессы о тех или иных событиях, и процессы могут сами решать, как на них реагировать.

В следующем разделе мы подробно рассмотрим, как работать с сигналами в Linux.

Сигналы в Linux

Сигналы обычно оповещают процесс Linux о том или ином событии. Например, ядро сигнализирует о сбое сегментации с помощью сигнала SIGSEGV, а если пользователь нажимает **Ctrl + C**, то выполнение текущей команды прерывается сигналом SIGINT. Управление процессами в Linux основано на сигналах, и в ядре определено несколько десятков сигналов. Каждому сигналу соответствует код — целое положительное число.

Следующая команда перечисляет все сигналы, которые зарегистрированы в системе Linux:

```
kill -l
```

Соответствующий вывод уже встречался нам на с. 194. Для примера рассмотрим пару сигналов из него. Сигналу SIGHUP соответствует код 1, и когда сеанс терминала завершает работу, он передает этот сигнал всем своим дочерним про-

цессам. А сигнал **SIGKILL** (код 9) чаще всего используется, чтобы завершать процессы. Процессы могут по-своему обрабатывать большинство сигналов, за исключением **SIGKILL** (9) и **SIGSTOP** (19), которые всегда завершают или приостанавливают процесс соответственно.

Процессы могут обработать сигнал одним из следующих способов:

- Выполнить действие по умолчанию, которое связано с сигналом: например, приостановить процесс, завершить его, выгрузить дампы памяти процесса или ничего не делать.
- Выполнить то или иное особое действие (за исключением сигналов **SIGKILL** и **SIGSTOP**). В этом случае процесс перехватывает сигнал и обрабатывает его по-своему.

Когда программа реализует собственную обработку сигнала, она обычно определяет функцию-обработчик, которая вмешивается в выполнение процесса по такой схеме:

- Получив сигнал, процесс прерывает свое выполнение на текущей инструкции.
- Управление немедленно переходит к функции-обработчику сигнала.
- Выполняется функция-обработчик.
- Когда функция-обработчик возвращает управление, процесс возобновляет выполнение, начиная с прерванной инструкции.

Процесс может проигнорировать сигнал, если у него есть соответствующий обработчик типа **NOOP** (ничего не делать). Из всех сигналов только два — **SIGKILL** и **SIGSTOP** — нельзя перехватить или проигнорировать.

Давайте рассмотрим несколько примеров того, как обрабатываются сигналы:

- Когда ядро возбуждает сигналы **SIGKILL**, **SIGFPE** (ошибочная арифметическая операция), **SIGSEGV** (ошибка сегментации), **SIGTERM** и им подобные, то процесс, который получил сигнал, обычно немедленно завершает выполнение и может создать дампы памяти — образ процесса, который используется для отладки.
- Когда пользователь нажимает **Ctrl + C**, процессу переднего плана передается сигнал **SIGINT** (запрос на прерывание). Процесс завершается, если только в соответствующей программе не реализован специальный обработчик **SIGINT**.
- Команда **kill** позволяет передать сигнал любому процессу, обратившись к нему по **PID**. Следующая команда передает сигнал **SIGHUP** сеансу терминала с **PID 3741**:

```
kill -HUP 3741
```

В командах такого рода можно указать либо код сигнала (например, 1 для **SIGHUP**), либо его имя без префикса **SIG** (например, **HUP** для **SIGHUP**).

С помощью `killall` можно передать один и тот же сигнал нескольким процессам, которые выполняют определенную команду (например, `test.sh`). Следующая команда завершает все процессы, которые выполняют сценарий `test.sh`, и выводит результат на консоль (с помощью ключа `-e`):

```
killall -e -TERM test.sh
```

Вывод этой команды показан ранее на с. 195.

Процессы и сигналы Linux — это обширная область, и здесь мы рассмотрели только самые элементарные ее понятия. Надеемся, что этот краткий обзор с практическими примерами наиболее типичных команд вдохновит вас на то, чтобы глубже погрузиться в эту тему и решать более сложные задачи.

Итоги

На то, чтобы изучить процессы и демоны Linux во всех подробностях, может понадобиться немало времени и усилий. Эта глава не претендует на то, чтобы соперничать с многочисленными томами специализированной литературы на эту тему. Тем не менее здесь мы попытались рассмотреть основные понятия в практическом духе, с примерами из реальной практики, чтобы компенсировать возможные упущения нестрогого изложения.

Мы надеемся, что к этому моменту вы уже освоились с процессами и демонами. Скорее всего, вы довольно хорошо ориентируетесь в типах процессов и в том, как они устроены, а также в общих чертах понимаете атрибуты и состояния процессов. Мы уделили особое внимание механизмам межпроцессного взаимодействия — в частности, сигналам. В главе 8 мы рассмотрим все эти темы еще подробнее. Думаем, что материала, который мы уже предоставили, должно хватить, чтобы понимать, как процессы обмениваются данными.

В следующей главе мы углубимся в то, как работать с дисками и файловыми системами Linux. Мы изучим, как хранятся данные в Linux, а также научимся сегментировать диски и управлять логическими томами. Не сомневайтесь, что все, что вы узнали до сих пор, будет то и дело пригождаться в последующих главах.

Вопросы

Если вы случайно пропустили что-то из материала этой главы, то, возможно, вам стоит освежить в памяти некоторые особенности процессов и демонов Linux:

1. Перечислите несколько типов процессов. В чем разница между ними?
2. Из чего состоит процесс? Назовите несколько основных атрибутов процесса (или полей в выводе `ps`), которые могут понадобиться, когда вы наблюдаете за процессами.

Подсказка: например, какие свойства процесса могут оказаться важными, помимо потребления процессора, оперативной памяти или диска?

3. Можете ли вы перечислить несколько состояний процесса и привести примеры того, как процессы переходят из одних состояний в другие?
4. Как узнать, какой процесс потребляет больше всего ресурсов центрального процессора в вашей системе?
5. Напишите простой сценарий и запустите его как долговременный фоновый процесс.

Подсказка: загляните в главу 8, которая посвящена тому, как создавать и использовать сценарии оболочки.

6. Перечислите не менее четырех сигналов, которые можно передать процессу. Когда или как вызываются эти сигналы?

Подсказка: используйте команду `kill -l`. Чтобы получить дополнительную информацию, обратитесь к справочным страницам.

7. Какие механизмы IPC вы знаете? Назовите плюсы и минусы каждого механизма.

Подсказка: вам может помочь материал из главы 8.

Дополнительная литература

Чтобы подробнее узнать обо всем, что рассматривалось в этой главе, можно обратиться к следующим изданиям Packt:

- Scott Alan Miller (Скотт Алан Миллер). «Linux Administration Best Practices».
- Donald A. Tevault (Дональд А. Тиволт). «Linux Service Management Made Easy with system».

Часть 2

РАСШИРЕННЫЕ ВОЗМОЖНОСТИ АДМИНИСТРИРОВАНИЯ LINUX

Во второй части этой книги вы узнаете о продвинутых задачах системного администрирования Linux — в частности, о том, как работать с дисками, настраивать сетевые службы, обеспечивать безопасность Linux, а также искать и диагностировать неисправности.

Эта часть состоит из следующих глав:

- *Глава 6. Как работать с дисками и файловыми системами*
- *Глава 7. Сетевое окружение Linux*
- *Глава 8. Сценарии командной оболочки в Linux*
- *Глава 9. Как обеспечивать безопасность в Linux*
- *Глава 10. Аварийное восстановление, диагностика и устранение неполадок*

6

Как работать с дисками и файловыми системами

В этой главе вы узнаете, как работать с дисками и файловыми системами, как управлять логическими томами, как монтировать и сегментировать дисковые накопители, а также получите представление о том, как хранятся данные в Linux. Кроме того, вы научитесь сегментировать и форматировать диски и создавать логические тома, а также подробнее узнаете о том, какие бывают файловые системы. В этой главе мы рассмотрим такие темы:

- Устройства в Linux.
- Файловые системы в Linux.
- Диски и разделы.
- Как управлять логическими томами.

Технические требования

Чтобы лучше усвоить материал этой главы, рекомендуется иметь общее представление о дисках, разделах и файловых системах. Кроме того, как и в предыдущих главах, вам понадобится действующая система Linux. Упражнения из этой главы мы будем демонстрировать в основном на примере Ubuntu или Debian, но все команды можно воспроизвести и в любом другом дистрибутиве Linux.

Устройства в Linux

Как уже неоднократно говорилось в этой книге, в Linux все является файлом, и устройства — не исключение. **Файлы устройств** — это специальные файлы в операционных системах семейства Unix и Linux. Эти файлы служат интерфейсами для драйверов устройств и представлены в файловой системе как обычные файлы.

Первым делом давайте посмотрим, как устроены уровни абстракции Linux. Это позволит вам лучше представить себе, как аппаратное и программное обеспечение связаны между собой и как они взаимодействуют друг с другом.

Уровни абстракции Linux

Сейчас самое время обсудить, как устроены уровни абстракции в Linux и как в эту схему вписываются устройства. Любой компьютер обычно состоит из аппаратного и программного обеспечения:

- **Аппаратное обеспечение.** К этой категории относятся аппаратные компоненты компьютера, такие как оперативная память, центральный процессор и прочее оборудование, в том числе диски, сетевые интерфейсы, порты и контроллеры.
- **Программное обеспечение.** Чтобы управлять всеми этими аппаратными компонентами, операционная система (в нашем случае Linux) использует **уровни абстракции**. Они реализуются в **ядре** — основном программном компоненте Linux. Если не углубляться в подробности, достаточно представить себе, что в Linux эти уровни отвечают за доступ к низкоуровневым ресурсам и обеспечивают необходимые драйверы для различных аппаратных компонентов.
- Когда компьютер включается, ядро Linux загружается с диска в оперативную память системы. При этом в памяти выделяются две отдельные области:
 - **Пространство ядра** предназначено для ядра ОС, которое находится в памяти и управляет всеми аппаратными компонентами. Оно служит интерфейсом между программным и аппаратным обеспечением Linux.
 - **Пользовательское пространство** — это уровень, на котором выполняются пользовательские процессы. Как говорилось в главе 5 «Как работать с процессами, демонами и сигналами», процесс — это запущенный экземпляр программы.

Где находятся устройства в этой всеобъемлющей схеме? За них отвечает ядро. Если говорить кратко, то ядро контролирует процессы, системные вызовы, память и устройства. Когда речь идет об устройствах, ядро управляет **драйверами устройств**, которые являются интерфейсом между оборудованием и программным обеспечением. Все устройства доступны только в режиме ядра, что позволяет обеспечить безопасность и оптимизировать работу.

Как все это функционирует? Оперативная память состоит из ячеек, в которых можно временно хранить данные. Программы, которые выполняются на компьютере, обращаются к этим ячейкам, и память служит посредником между процессором и постоянным хранилищем. Доступ к памяти происходит очень быстро, благодаря чему программы могут выполняться бесперебойно. Управлять пользовательскими процессами в пользовательском пространстве — задача ядра, которое следит за тем, чтобы ни один из процессов не мешал другим. Доступ к пространству ядра обычно есть только у ядра, но бывают случаи, когда этот доступ нужен пользовательским процессам. Для этого служат **системные вызовы**. Системный вызов работает так: пользовательский процесс запрашивает службу ядра, обращаясь к нужным ресурсам через активный процесс в про-

странстве ядра — например, чтобы передать запросы ввода-вывода к внутренним или внешним устройствам. Чтобы выполнить нужные операции, все эти запросы обмениваются данными с процессором через оперативную память.

В следующем разделе мы узнаем, как управлять файлами устройств в Linux и как их принято именовать.

Файлы устройств и соглашения об именах

После того как вы узнали, как устроены уровни абстракции, вам может быть интересно, как Linux управляет устройствами. Это происходит с помощью **пользовательского пространства** `/dev` (`udev`), которое служит диспетчером устройств для ядра. Оно имеет дело с **узлами устройств** — это специальные файлы (также называемые **файлами устройств**), которые используются в качестве интерфейса для драйверов.

Файлы устройств в Linux

`udev` работает как демон, прослушивая вызовы, которые ядро передает в пользовательское пространство, поэтому он осведомлен о том, какие типы устройств используются и каким образом. Соответствующий демон называется `udev`, и его конфигурация в современных системах находится в файле `/etc/udev/udev.conf`. Чтобы увидеть его содержимое, можно запустить такую команду:

```
cat /etc/udev/udev.conf
```

В каждом дистрибутиве Linux есть стандартный набор правил, которые регулируют работу `udev`. Они обычно хранятся в каталоге `/etc/udev/rules.d/`:

```
packt@neptune:~$ ls -l /etc/udev/rules.d/
total 68
-rw-r--r-- 1 root root 63312 Aug  9 2022 70-snap.snapd.rules
-rw-r--r-- 1 root root  148 Feb 27 08:50 ubuntu--vq-ubuntu--lv.rules
```

Ядро транслирует события с помощью сокета **Netlink**. Это интерфейс для меж-процессного взаимодействия, к которому обращаются как процессы пользовательского пространства, так и процессы пространства ядра.

Каталог `/dev` — это интерфейс между пользовательскими процессами и устройствами под управлением ядра. Если вы запустите команду `ls -la /dev`, то увидите в этом каталоге множество файлов с разнообразными именами, которые относятся к разным типам. Некоторые файлы помечены буквами *b* и *c*, однако могут встречаться также буквы *p* и *s*. Файлы соответствующих типов — это файлы устройств. Буква *b* обозначает **блочные устройства**, а *c* — **символьные**:

```
packt@neptune:~$ ls -la /dev
total 4
```

```

drwxr-xr-x 20 root root          4080 Mar 16 07:09 .
drwxr-xr-x 19 root root          4096 Feb 27 08:54 ..
crw-r--r--  1 root root         10, 235 Mar 16 07:09 autofs
drwxr-xr-x  2 root root          360 Mar 16 07:09 block
crw-rw----  1 root disk        10, 234 Mar 16 07:09 btrfs-control
drwxr-xr-x  3 root root          60 Mar 16 07:09 bus
drwxr-xr-x  2 root root        3500 Mar 16 07:09 char
crw--w----  1 root tty          5,   1 Mar 16 07:09 console
lrwxrwxrwx  1 root root          11 Mar 16 07:09 core -> /proc/kcore
drwxr-xr-x  3 root root          60 Mar 16 07:09 cpu
crw-----  1 root root        10, 124 Mar 16 07:09 cpu_dma_latency
crw-----  1 root root        10, 203 Mar 16 07:09 cuse
drwxr-xr-x  7 root root          140 Mar 16 07:09 disk
brw-rw----  1 root disk       253,   0 Mar 16 07:09 dm-0
drwxr-xr-x  2 root root          60 Mar 16 07:09 dma_heap
drwxr-xr-x  3 root root          80 Mar 16 07:09 dri
crw-----  1 root root        10, 126 Mar 16 07:09 ecryptfs
crw-rw----  1 root video       29,   0 Mar 16 07:09 fb0
lrwxrwxrwx  1 root root          13 Mar 16 07:09 fd -> /proc/self/fd

```

Давайте посмотрим, как в каталоге `/dev` представлены дисковые устройства. Но сначала скажем несколько слов о системах, на примере которых мы демонстрируем упражнения.

В большинстве упражнений из этой книги мы используем операционные системы семейства Linux, которые работают в виртуальных машинах, названных в честь планет Солнечной системы. Например, сервер Ubuntu 22.04.2 LTS работает под управлением машины `neptune`, поэтому если вы видите это имя в приглашении оболочки, то можете сделать вывод, что мы имеем дело с Ubuntu. Вот полный список наших виртуальных машин:

<code>neptune</code>	Ubuntu 22.04.2 LTS Server
<code>jupiter</code>	openSUSE 15.4 Leap (сервер)
<code>saturn</code>	Fedora 37 Workstation
<code>venus</code>	AlmaLinux
<code>mars</code>	Debian 11.6 (сервер)

В системе под управлением виртуальной машины драйверы устройств называются не так, как на физическом оборудовании. Мы расскажем об этом подробнее в следующем разделе, когда будем рассматривать соглашения об именовании устройств. Впрочем, для некоторых примеров, которые специально отмечены, мы используем свою основную рабочую станцию под управлением Debian 12 GNU/Linux.

Как показано в примере ниже, дисковый накопитель `sda` представлен в виде блочного устройства. Блочные устройства имеют фиксированный размер, и их содержимое легко индексировать. А у символьных устройств нет определенно-

го размера, и к ним можно обращаться с помощью потоков данных. Например, принтеры представляются как символьные устройства. Здесь мы использовали основную рабочую станцию под управлением Debian GNU/Linux, где `sda1` — это внешнее устройство USB:

```
brw-rw---- 1 root disk 8, 0 Mar 16 09:40 sda
brw-rw---- 1 root disk 8, 0 Mar 16 09:40 sda1
crw-rw---- 1 root disk 21, 0 Mar 16 09:40 sda
```

Для сравнения: если мы перечислим устройства на виртуальной машине `neptune`, то получим такой результат:

```
brw-rw---- 1 root disk 252, 0 Mar 16 07:09 vda
brw-rw---- 1 root disk 252, 1 Mar 16 07:09 vda1
brw-rw---- 1 root disk 252, 2 Mar 16 07:09 vda2
brw-rw---- 1 root disk 252, 3 Mar 16 07:09 vda3
```

Блочные устройства, которые обозначены как `vdaX`, — это виртуальные устройства внутри виртуальной машины. Подробнее о виртуальных машинах вы узнаете в главе 11 «Как работать с виртуальными машинами».

А сейчас давайте поговорим о соглашениях об именах устройств в Linux.

Соглашения об именах устройств в Linux

В Linux используются соглашения об именах устройств, благодаря которым управлять устройствами становится проще, и во всей экосистеме Linux поддерживается общий порядок. В `udev` используется несколько специальных схем именования, которые по умолчанию присваивают устройствам фиксированные имена, причем эти имена стандартизированы в зависимости от категории устройств. Например, когда ядро присваивает имена сетевым устройствам, оно использует данные из таких источников, как версия прошивки, топология и местоположение. В прежних системах на основе Red Hat использовалось пять схем именования сетевых интерфейсов, с которыми можно ознакомиться в официальной документации: docs.redhat.com/en/documentation/red_hat_enterprise_linux/7/html/networking_guide/ch-consistent_network_device_naming¹.

В системах семейства Debian применяется похожая схема именования, которая обеспечивает предсказуемость благодаря тому, что основана на именах аппаратных шин. Это характерно для всех современных операционных систем на основе Linux.

Если вы хотите посмотреть, какие правила `udev` активны в вашей системе, имейте в виду, что в дистрибутивах семейств Debian и Red Hat они хранятся в каталоге `/lib/udev/rules.d/`.

¹ Начиная с RHEL 9, в этом дистрибутиве по умолчанию применяется только одна схема именования сетевых интерфейсов, которая описана на странице docs.redhat.com/en/documentation/red_hat_enterprise_linux/9/html/configuring_and_managing_networking/consistent-network-interface-device-naming_configuring-and-managing-networking. — *Примеч. ред.*

Для жестких дисков или внешних накопителей применяются более единообразные схемы именования. Вот несколько примеров:

Классические драйверы IDE для дисков ATA	
<code>hda</code>	ведущее устройство
<code>hdb</code>	ведомое устройство на первом канале
<code>hdc</code>	ведущее устройство на втором канале
<code>hdd</code>	ведомое устройство на втором канале
Драйверы NVMe	
<code>nvme0</code>	первый контроллер устройства (символьное устройство)
<code>nvme0n1</code>	первое пространство имен (блочное устройство)
<code>nvme0n1p1</code>	первое пространство имен, первый раздел (блочное устройство)
Драйверы SD-карт (MMC)	
<code>mmcblk</code>	SD-карты с чипами eMMC
<code>mmcblk0</code>	первое устройство
<code>mmcblk0p1</code>	первое устройство, первый раздел
Драйверы SCSI для современных контроллеров SATA или USB	
<code>sd</code>	устройства хранения данных
<code>sda</code>	первое зарегистрированное устройство
<code>sdb (sdc, sdd и т. д.)</code>	второе (третье, четвертое и т. д.) зарегистрированное устройство
<code>sg</code>	уровень SCSI общего вида (символьное устройство)

В этой главе нас больше всего интересуют устройства хранения данных — обычно это жесткие диски (HDD) или твердотельные накопители (SSD), которые чаще всего используются в компьютерах. Как правило, эти диски разбиты на разделы с определенной структурой, которую обеспечивает файловая система. Мы уже немного говорили о файловых системах в главе 2 «Командная оболочка и файловая система Linux», когда упоминали структуру каталогов, но теперь пришло время подробнее рассказать о том, какие файловые системы применяются в Linux.

Файловые системы в Linux

Когда мы говорим о физических носителях, таких как жесткие диски или внешние накопители, мы не имеем в виду иерархию каталогов. Вместо этого речь идет о структурах, которые возникают на физическом диске, когда его форма-

тируют и/или разбивают на разделы. Эти структуры называются файловыми системами, и они определяют, как работать с файлами, когда они хранятся на диске.

Существуют разные файловые системы: одни из них органически развивались в экосистеме Linux, а другие — нет (например, специфические файловые системы Windows или macOS). В этом разделе мы опишем только «родные» файловые системы Linux.

В Linux наиболее распространены системы семейства ext (ext, ext2, ext3 и ext4), а также XFS, ZFS и Btrfs. У каждой из них есть сильные и слабые стороны, но все они неплохо справляются с задачами, для которых создавались. Системы семейства ext используются особенно широко и за долгие годы доказали свою надежность. ext4 — последняя версия, которая похожа на ext3, но лучше поддерживает большие файлы, а также оптимизирована с точки зрения фрагментации и производительности. ext3 использует 32-битную адресацию, а ext4 — 48-битную, что позволяет поддерживать файлы объемом до 16 Тбайт. Она также допускает неограниченное количество подкаталогов в каждом каталоге, в то время как в ext3 может быть только 32 000 подкаталогов. Кроме того, в ext4 поддерживается «горячая» дефрагментация на уровне ядра, а временные метки расширены на два бита, что позволяет хранить даты вплоть до 2446 года нашей эры.

Тем не менее ext4 — это, строго говоря, не файловая система нового поколения; скорее это улучшенная, надежная, прочная и стабильная историческая реликвия, которая не прошла тест на безопасность и целостность данных. Ее механизм протоколирования не позволяет обнаруживать и устранять повреждения и деградацию. Именно поэтому другие файловые системы, такие как XFS и ZFS, начали возрождаться соответственно в Red Hat Enterprise Linux, начиная с версии 7, и в Ubuntu, начиная с версии 16.04.

Btrfs — это особый случай. Она считается современной файловой системой, но все еще используется как однодисковая система и не применяется в многодисковых системах управления томами из-за ряда проблем с производительностью по сравнению с другими файловыми системами. Она по умолчанию используется в настольных версиях Fedora, начиная с версии 33, а также доступна в SUSE Linux Enterprise и openSUSE, однако больше не поддерживается в Red Hat.

Вот некоторые подробности об основных файловых системах:

- **ext4.** Файловая система ext4 с самого начала разрабатывалась для Linux. Несмотря на то что ее постепенно вытесняют другие системы, она обладает внушительными возможностями. В ней можно выбрать размер блока (от 512 до 4096 байт), а также поддерживается резервирование индексных дескрипторов, которое экономит их при создании каталогов и таким образом повышает производительность при создании новых файлов.

Байты файлов следуют в «остроконечном» порядке (от младшего к старшему, подробнее об этом можно узнать на странице cloudzilla.ai/dev-education/what-is-little-endian-and-big-endian), а группы блоков содержат данные индексных дескрипторов, что позволяет быстрее обращаться к файлам. Для каждого файла предварительно выделяются блоки данных, благодаря чему уменьшается фрагментация. Кроме того, в ext4 реализовано много усовершенствований, например: она поддерживает максимальный размер файловой системы в 1 экзбайт (1 Эбайт) и многоблочное распределение, может разбивать большие файлы на фрагменты максимально возможного объема и применяет отложенное выделение места, чтобы повысить производительность. Также ext4 позволяет быстро проверять файловую систему с помощью удобной команды `fsck`, использует контрольные суммы, чтобы улучшить надежность системы и качество протоколирования, а также поддерживает улучшенные временные метки.

- **ZFS.** Эта файловая система была создана в компании Sun Microsystems и объединяет в себе файловую систему и систему управления логическими томами. Ее начали разрабатывать в 2001 году, а анонсировали в 2004-м. Сначала она была интегрирована в операционную систему Solaris, а затем использовалась (хотя и не по умолчанию) в Debian, FreeBSD и других ОС. ZFS — это хорошо масштабируемая система со 128-битной адресацией, которая обеспечивает простое администрирование, целостность данных, масштабируемость и производительность. Разработка этой файловой системы ведется в рамках проекта с открытым исходным кодом **OpenZFS**. ZFS предлагает сложную структуру данных с механизмом копирования при записи, что отличается от традиционных файловых систем. Чтобы подробнее узнать о ZFS, мы рекомендуем официальный справочник: [openzfs.github.io/openzfs-docs/Getting Started](https://openzfs.github.io/openzfs-docs/Getting%20Started).
- **XFS.** Корпоративные версии Linux начинают меняться, переходя с ext4 на другие конкурентоспособные файловые системы. Среди них — XFS, которая была создана компанией Silicon Graphics и использовалась в операционной системе IRIX. Важнейшее преимущество XFS — производительность, поскольку эта файловая система способна работать с большими массивами данных. Кроме того, она позволяет выполнять параллельные задачи ввода-вывода с гарантированно высокой скоростью передачи данных. Файловая система поддерживает объем хранилища до 16 Эбайт, а размер отдельного файла может достигать 8 Эбайт. XFS умеет протолировать информацию о квотах, а также поддерживает обслуживание в «горячем» режиме, в том числе дефрагментацию, расширение и восстановление. Также предусмотрены специальные инструменты для резервного копирования и восстановления, включая `xfsdump` и `xfsrestore`.
- **Btrfs.** Эта файловая система все еще находится в стадии разработки, но она решает многие проблемы существующих систем, поддерживая моментальные снимки, объединение носителей, контрольные суммы и размещение на не-

скольких устройствах — все эти функции необходимы в корпоративной среде.

Существуют и другие файловые системы, которые здесь подробно не рассматриваются: **ReiserFS**, **GlusterFS**, **NFS**, **SMB**, **ISO9660** для CD-ROM и расширений Joliet, а также «родные» системы других ОС — такие, как **FAT**, **NTFS**, **exFAT**, **APFS**, **HFS+** и др. Если вы хотите больше узнать о них, не стесняйтесь самостоятельно искать информацию; хорошей отправной точкой будет «Википедия»: en.wikipedia.org/wiki/File_system. Чтобы узнать, какие файловые системы поддерживаются в вашем дистрибутиве Linux, запустите команду `cat /proc/filesystems`.

В Linux реализовано особое программное решение, которое способно выполнять определенные функции файловых систем. Оно известно как **виртуальная файловая система** и служит связующим звеном между ядром, различными типами файловых систем и аппаратным обеспечением. Поэтому, когда приложение хочет открыть файл, эта операция выполняется через виртуальную файловую систему как слой абстракции:

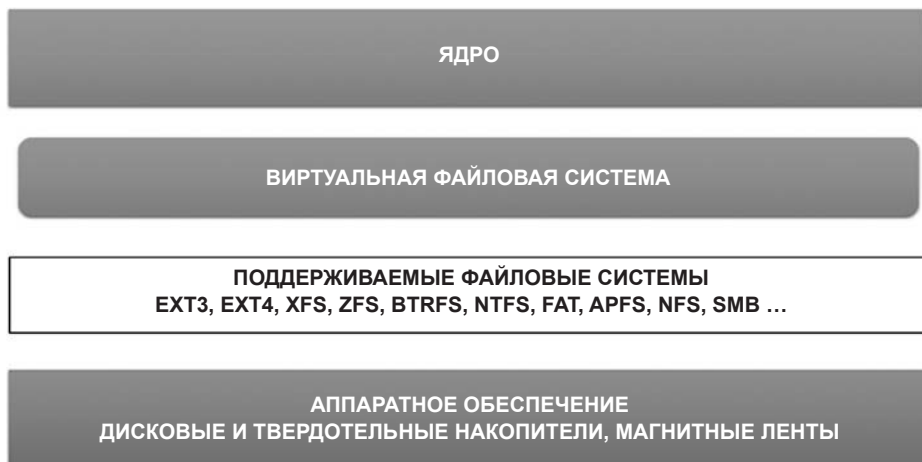


Рис. 6.1. Виртуальная файловая система Linux как слой абстракции

Основные функции файловой системы заключаются в том, чтобы создавать пространства имен, поддерживать структуры метаданных, которые формируют логическую основу для иерархических структур каталогов, управлять использованием дисковых блоков, хранить информацию о размере файлов и доступе к ним, а также высокоуровневые данные для логических томов и разделов. Кроме того, у каждой файловой системы есть свой **API**, благодаря которому у разработчиков появляется доступ к вызовам системных функций, которые манипулируют объектами файловой системы и реализуют определенные алгоритмы для того, чтобы создавать, перемещать и удалять файлы, а также индек-

сировать и искать их. Кроме того, каждая современная файловая система поддерживает особую схему прав доступа, с помощью которой задаются правила доступа пользователей к файлам.

Итак, мы рассмотрели основные файловые системы Linux, включая ext4, Btrfs и XFS. В следующем разделе мы расскажем, как управлять дисками и разделами в Linux.

Диски и разделы

Любому системному администратору важно разбираться в том, как устроены диски и разделы, а также уметь грамотно форматировать и сегментировать диски, начиная с этапа установки системы. Не менее важно представлять себе, какие аппаратные компоненты доступны в вашей системе и как с ними работать. Один из этих компонентов — диск, и сейчас мы поговорим о нем подробнее.

Распространенные типы дисков

Диск — это аппаратное устройство, на котором хранятся данные. Диски бывают разных типов и используют различные интерфейсы. К основным типам дисков относятся традиционные магнитные диски и накопители SSD (которые подключаются, в частности, через интерфейсы SATA и NVMe). Технически SSD во многом похожи на оперативную память, благодаря чему эффективнее потребляют энергию и быстрее передают и принимают данные, чем вращающиеся жесткие диски. Чтобы обмениваться данными с дисковыми устройствами, используются такие интерфейсы:

- **IDE.** Это устаревший стандарт, который до сих пор иногда применяется в оборудовании потребительского сегмента с небольшой скоростью передачи данных.
- **SATA.** Пришел на смену IDE и передает данные со скоростью до 16 Гбит/с.
- **SCSI.** Используется в основном в серверах корпоративного уровня, где применяется RAID со сложной аппаратной конфигурацией.
- **SAS.** Это последовательный интерфейс типа «точка-точка», который передает данные примерно с такой же скоростью, как SATA. Благодаря своей надежности он чаще всего используется в корпоративных средах.
- **USB.** Позволяет подключать внешние жесткие диски и накопители.

Чтобы просмотреть сведения о таблицах разделов диска, в Linux можно запустить команду `fdisk -l`.

На нашей основной рабочей станции под управлением Debian 12 GNU/Linux есть один накопитель SSD, а к одному из портов USB подключено внешнее устройство. Вот что выводит эта команда в нашем случае:

```
alexandru@debian:~$ sudo fdisk -l
Disk /dev/nvme0n1: 953.87 GiB, 1024209543168 bytes, 2000409264 sectors
Disk model: Lexar SSD NM620 1TB
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: gpt
Disk identifier: A6049E7C-127A-40D3-9D3F-F5CF9D463B88
```

Device	Start	End	Sectors	Size	Type
/dev/nvme0n1p1	2048	1050623	1048576	512M	EFI system
/dev/nvme0n1p2	1050624	2050047	999424	488M	Linux filesystem
/dev/nvme0n1p3	2050048	2000408575	1998358528	952.9G	Linux filesystem

```
Disk /dev/sda: 7.5 GiB, 8053063680 bytes, 15728640 sectors
Disk model: Flash Disk
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x24248bcf
```

Device	Boot	Start	End	Sectors	Size	Id	Type
/dev/sdal		2048	15728639	15726592	7.5G	83	Linux

Этот вывод может ужаснуть с непривычки, но будьте уверены, что мы объясним его так, чтобы он выглядел гораздо дружелюбнее. Если запустить `fdisk`, не указав конкретного раздела в качестве аргумента, то будет выведена вся информация о разделах, которые перечислены в файле `/proc/partitions`. В предыдущем примере отображаются сведения о двух дисках в нашей системе: один из них — твердотельный накопитель Lexar NM620 емкостью 1 Тбайт, а другой — накопитель USB емкостью 8 Гбайт. Давайте разберемся, что система говорит о накопителе SSD:

- Во-первых, поле `Disk model` (Модель диска) сообщает, как называется диск, а поле `Units` (Блоки) указывает на то, что минимальной адресуемой единицей хранения информации на нем является сектор размером 512 байт. `Disklabel type` (Тип метки диска) — тип таблицы разделов диска (в данном случае GPT), а `Disk identifier` — идентификатор, который уникален для каждого диска.
- Далее находится таблица разделов, которые доступны на диске. В этой таблице шесть столбцов (а в случае с накопителем USB их семь). Первый столбец `Device` (Устройство) содержит имена разделов в соответствии с текущей схемой именования. Во втором и третьем столбцах указаны адреса начального и конечного секторов, а в четвертом столбце — общее количество секторов в разделе. Пятый столбец (`Size`) содержит объем раздела в человеко-читаемом формате, а шестой (`Type`) — тип файловой системы.

Владеть основными сведениями о дисковых устройствах в вашей системе — это лишь первый шаг в работе с дисками и разделами в Linux. Если не отформати-

ровать диск и не разбить его на разделы, которые система сможет использовать, он останется просто куском металла. Именно поэтому далее мы расскажем, что такое разделы.

Как разбивать диски на разделы

Обычно диски разбиваются на **разделы**. Чаще всего разделы — это смежные блоки физических единиц хранения информации, причем разделы бывают **первичными, расширенными и логическими**. Linux распознает не более 15 разделов на одном диске, причем из четырех первых разделов только один может быть первичным или расширенным, а остальные три — только первичными; прочие разделы могут быть только логическими. Впрочем, единственный расширенный раздел можно разбить на несколько логических, если не превышать общее допустимое количество разделов.

Форматы сегментации диска

Существует два основных формата сегментации — **MBR** и **GPT**¹. MBR интенсивно использовался примерно до 2010 года и поддерживал не более четырех первичных разделов при максимальном объеме раздела 2 Тбайт. В MBR различные типы разделов обозначаются шестнадцатеричными кодами: например, FAT-32X — 0x0c, NTFS и некоторые другие типы — 0x07, разделы данных Linux — 0x83, а раздел подкачки Linux — 0x82. GPT — часть стандарта **UEFI**; этот формат был разработан для того, чтобы преодолеть некоторые проблемы MBR, в том числе ограничения на количество разделов, несовершенные методы адресации, поддержку только одной копии таблицы разделов и т. д. Формат GPT поддерживает до 128 разделов и накопители размером до 75,6 зеттабайт (Збайт).

¹ MBR расшифровывается как Master Boot Record (главная загрузочная запись), а GPT — как GUID Partition Table (таблица разделов GUID). В свою очередь, GUID — это Globally Unique Identifier (глобально уникальный идентификатор). Стоит отметить, что названия и аббревиатуры многих технических решений не отражают их сути и могут даже дезинформировать (как, например, название команды `kill`, которая применяется далеко не только для того, чтобы «убить» процесс). Бывает, что технологии развиваются так, что со временем перестают соответствовать своему изначальному названию, однако по тем или иным причинам оно сохраняется: например, протокол HTTP (протокол передачи гипертекста) не переименовали после того, как он научился передавать не только документы HTML, но и любые другие данные. В других случаях термины с самого начала оказывались не вполне удачными, потому что их авторы в первую очередь заботились о технических качествах своих разработок, а не об идеальном нейминге. Профессиональный подход состоит в том, чтобы понимать, как на самом деле устроены те или иные компоненты и как они работают на практике, а не только как они официально называются. — *Примеч. ред.*

Таблица разделов

Таблица разделов диска описывает его структуру разделов и хранится на выделенном участке этого диска¹. Например, запись MBR занимает первые 512 байт диска, из которых под таблицу разделов отведено 64 байта почти в самом конце записи. (Еще 2 байта после таблицы разделов известны как сигнатура загрузочного раздела.) Первые 446 байт MBR зарезервированы для кода, который обычно относится к программе-загрузчику (по крайней мере в классических версиях MBR). В случае Linux загрузчик называется **GRUB**.

Когда система Linux загружается, загрузчик ищет активный раздел: на каждом диске может быть только один такой раздел. Когда он найден, загрузчик загружает нужные компоненты. Таблица разделов MBR состоит из 4 записей, каждая из которых занимает 16 байт и соответствует одному из возможных основных разделов системы. Кроме того, каждая запись содержит метаданные — в частности, начальный и конечный адреса раздела в формате CHS (*цилиндр/головка/сектор*), код типа раздела и количество секторов в разделе.

Имена разделов

Ядро взаимодействует с диском на низком уровне, используя при этом узлы устройств, которые хранятся в каталоге `/dev`; примеры его содержимого уже демонстрировались ранее в этом разделе (с. 213 и 214). В случае устройств SCSI или SATA диски и их разделы обозначаются так:

- первый диск всегда обозначается `/dev/sda`;
- второй диск — `/dev/sdb`, третий — `/dev/sdc` и т. д.;
- первый раздел первого диска — `/dev/sda1`;
- первый раздел второго диска — `/dev/sdb1`;
- второй раздел второго диска — `/dev/sdb2` и т. д.

Ядро присваивает дискам буквенные обозначения (*a, b, c* и т. д.), опираясь на идентификационный номер устройства, а не на то, где оно находится на аппаратной шине.

Атрибуты разделов

Чтобы просмотреть атрибуты раздела, можно воспользоваться командой `lsblk`. Вот что она выводит в нашей системе Debian:

```
alexandru@debian:~$ lsblk
NAME                                MAJ:MIN RM  SIZE RO TYPE  MOUNTPOINTS
```

¹ Внутреннее устройство таблицы разделов в книге описывается в основном в соответствии с классической схемой MBR. Узнать подробнее о GPT можно на сайте Родерика Смита (Roderick W. Smith), создателя специализированных утилит для работы с GPT: rodsbooks.com/gdisk. — *Примеч. ред.*


```
sda            8:0      1    7.5G  0 disk
└─sda1         8:1      1    7.5G  0 part  /media/alexandru/9d7587da-3e17-464c-
b01dd044b82f3736
nvme0n1        259:0      0 953.9G  0 disk
├─nvme0n1p1    259:1      0    512M  0 part  /boot/efi
├─nvme0n1p2    259:2      0    488M  0 part  /boot
├─nvme0n1p3    259:3      0 952.9G  0 part
└─nvme0n1p3_crypt
    254:0        0 952.9G  0 crypt
    └─debian--vg-root
        254:1      0 951.9G  0 lvm    /
    └─debian--vg-swap_1
        254:2      0    976M  0 lvm    [SWAP]
```

Команда `lsblk` показывает имя устройства (имя узла из `sysfs` и базы данных `udev`), главный и дополнительный номер устройства, съемное ли оно (0 — несъемное, 1 — съемное), объем в человекочитаемом формате, режим доступа (1 — если устройство доступно только для чтения, 0 — в остальных случаях), тип устройства и точку его монтирования (если она доступна).

Теперь, когда мы больше знаем о дисках, давайте рассмотрим, как изменить таблицу разделов диска.

Как редактировать таблицы разделов

В Linux есть несколько инструментов, с помощью которых можно управлять таблицами разделов. Вот самые распространенные из них:

<code>fdisk</code>	Редактор разделов с интерфейсом командной строки; возможно, самый популярный
<code>sfdisk</code>	Неинтерактивный редактор разделов; используется в основном в коде сценариев
<code>parted</code>	Альтернативная программа для работы с разделами; распространяется в рамках проекта GNU
<code>GParted</code>	Графический интерфейс для <code>parted</code>

Далее мы подробно рассмотрим только `fdisk`, потому что это самый распространенный редактор разделов в Linux. Он есть как в Ubuntu/Debian, так и в RHEL/Fedora или openSUSE, а также во многих других дистрибутивах.

Но прежде чем запускать `fdisk`, давайте посмотрим, какие разделы распознает операционная система. Для этого можно вывести содержимое файла `/proc/partitions` с помощью команды `cat`:

```
alexandru@debian:~$ cat /proc/partitions
major minor #blocks name
259      0 1000204632 nvme0n1
259      1    524288 nvme0n1p1
```

```

259      2      499712 nvme0n1p2
259      3 999179264 nvme0n1p3
254      0 999162880 dm-0
254      1 998162432 dm-1
254      2   999424 dm-2
      8      0   7864320 sda
      8      1   7863296 sda1

```

Утилиту `fdisk` можно запускать только с правами суперпользователя. Мы советуем вам экспериментировать с ней осторожно, потому что она может повредить существующие разделы и диски. Так `fdisk` можно запустить для конкретного диска:

```
alexandru@debian:~$ sudo fdisk /dev/sda
```

```

Welcome to fdisk (util-linux 2.38.1).
Changes will remain in memory only, until you decide to write them.
Be careful before using the write command1.

```

```
Command (m for help)2: _
```

Как видите, `fdisk` предупреждает вас о том, что изменения применятся к диску только после того, как вы вызовете команду записи. Утилита также предлагает ввести команду: в частности, команда `m` позволяет получить справку. Мы советуем всегда сверяться со справкой, даже если вы уже знаете наиболее используемые команды.

Если ввести `m`, вы увидите полный список команд, которые доступны в `fdisk`: они позволяют управлять разделами, создавать новые загрузочные записи, сохранять изменения и т. д. Редакторы таблиц разделов — важные инструменты для управления дисками в Linux, но вы не сможете использовать их на полную мощность, если не узнаете, как форматировать разделы. В следующем разделе мы поговорим о том, как разбить диск на разделы.

Как создавать и форматировать разделы диска

В этом разделе мы с помощью `fdisk` создадим новую таблицу разделов на накопителе USB, который подключен к нашей основной рабочей станции под управлением Debian GNU/Linux. Чтобы войти в интерфейс `fdisk`, запустим уже знакомую команду:

```
sudo fdisk /dev/sda
```

Теперь введем команду `o`, чтобы создать пустую таблицу разделов MBR, а затем команду `w`, чтобы сохранить изменения на диске. Результат выглядит так:

¹ Добро пожаловать в утилиту `fdisk` (из пакета `util-linux 2.38.1`). Изменения будут сохраняться только в памяти, пока вы не вызовете команду записи. — *Примеч. пер.*

² Введите команду (чтобы вызвать справку, введите `m`). — *Примеч. пер.*

```
alexandru@debian:~$ sudo fdisk /dev/sda
```

```
Welcome to fdisk (util-linux 2.38.1).
Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.
```

```
Command (m for help): o
Created a new DOS (MBR) disklabel with disk identifier 0xc1fa6ca8.1
```

```
Command (m for help): w
The partition table has been altered.
Calling ioctl() to re-read partition table.
Syncing disks.
```

После этого таблица разделов создана, но ни одного раздела на диске еще нет. Находясь в интерфейсе `fdisk`, можно ввести команду `v`, чтобы проверить только что созданную таблицу, и команду `i`, чтобы просмотреть сведения о существующих разделах. Вы увидите сообщение о том, что разделы еще не определены. Значит, пришло время их определить.

Чтобы создать новый раздел, мы введем такую последовательность команд:

- `n`, чтобы начать создание раздела;
- `p` в ответ на запрос о том, какого типа должен быть раздел — основным (`p`) или расширенным (`e`)²;
- Номер раздела (по умолчанию `1`);
- Номер первого сектора (по умолчанию `2048`);
- Номер последнего сектора. Если вам нужен раздел определенного объема, его можно явно указать: например, `+2G` означает раздел объемом 2 Гбайт. По умолчанию создается раздел максимально возможного объема.
- Если команда предложит удалить какие-либо сигнатуры, введите `Y`, чтобы подтвердить это действие;
- `w`, чтобы записать изменения на диск.

Результат этих операций выглядит так:

```
Command (m for help): n
Partition type      # Тип раздела
  p   primary (0 primary, 0 extended, 4 free)
      # Первичный (создано 0 первичных и 0 расширенных разделов,
      # доступно 4 раздела)
  e   extended (container for logical partitions)
      # Расширенный (контейнер для логических разделов)
Select (default p): p      # Выберите тип раздела (по умолчанию p)
```

¹ Создана новая метка диска в формате DOS (MBR) с идентификатором диска `0xc1fa6ca8`. — *Примеч. пер.*

² На этом и последующих шагах можно ничего не вводить, а просто нажать `Enter`, если вы хотите применить значение по умолчанию (в данном случае `p`). — *Примеч. ред.*

```

Partition number (1-4, default 1): 1          # Номер раздела (1-4, по умолчанию
1)
First sector (2048-15728639, default 2048): # Первый сектор
Last sector, +/-sectors or +/-size{K,M,G,T,P} (2048-15728639, default 15728639):
# Последний сектор в формате ±кол-во_секторов или ±объем
Created a new partition 1 of type 'Linux' and of size 7.5 GiB.
# Создан новый раздел 1 типа Linux объемом 7,5 Гбайт
Partition #1 contains a ext4 signature.
# Раздел 1 содержит сигнатуру ext4

Do you want to remove the signature? [Y]es/[N]o: Y
# Хотите ли вы удалить эту сигнатуру?

The signature will be removed by a write command.
# Сигнатура будет удалена после вызова команды записи (w)

Command (m for help): w
The partition table has been altered.
# Таблица разделов изменена
Calling ioctl() to re-read partition table.
# Вызывается системный вызов ioctl(), чтобы заново считать таблицу разделов
Syncing disks.
# Диски синхронизируются

```

Таким образом, мы создали раздел, но еще не отформатировали его. Прежде чем форматировать, давайте узнаем, как создать резервную копию таблицы разделов.

Бывают ситуации, когда необходимо создать резервную копию, чтобы позже восстановить по ней таблицу разделов. Поскольку сегментация диска иногда может пойти не по плану, стоит придерживаться хорошей стратегии резервного копирования. В этом может помочь утилита `dd`, которую можно запустить так:

```
sudo dd if=/dev/sda of=mbr-backup bs=512 count=1
```

Это очень полезная и мощная программа: в частности, она умеет клонировать и очищать диски. Вот пример для нашего случая:

```

alexandru@debian:~$ pwd
/home/alexandru

alexandru@debian:~$ sudo dd if=/dev/sda
sda sda1

alexandru@debian:~$ sudo dd if=/dev/sda of=mbr-backup bs=512 count=1
1+0 records in
1+0 records out
512 bytes copied, 0.00138593 s, 369 kB/s

alexandru@debian:~$ ls
Desktop  Downloads  Music      Public     Videos
Documents mbr-backup Pictures   Templates

```

Синтаксис `dd` довольно прост. По умолчанию она принимает данные из стандартного входного потока и выводит результаты в стандартный выходной поток, но это можно изменить, если указать нужный входной файл с помощью ключа `if`, а выходной — с помощью ключа `of`¹. Здесь в качестве входного файла мы задали файл дискового устройства, для которого мы хотим сделать резервную копию (`/dev/sda`), а в качестве выходного файла — имя резервной копии (`mbr-backup`). Ключ `bs` обозначает размер каждого блока, а `count` определяет, сколько блоков будет скопировано.

Чтобы восстановить загрузчик, можно вызвать ту же команду `dd` похожим образом:

```
sudo dd if=~ /mbr-backup of=/dev/sda bs=512 count=1
```

Теперь, когда вы знаете, как создать резервную копию таблицы разделов с помощью `dd`, давайте отформатируем раздел, который мы создали ранее. Для форматирования чаще всего используют программу `mkfs`²: фактически это обертка, которая вызывает ту или иную конкретную утилиту в зависимости от файловой системы. Вот файловые системы, которые поддерживает `mkfs` (полный список зависит от дистрибутива и пользовательских настроек):

```
alexandru@debian:~$ ls -lh /sbin/mkfs*
-rwxr-xr-x 1 root root 15K Feb 13 10:48 /sbin/mkfs
-rwxr-xr-x 1 root root 35K Feb 13 10:48 /sbin/mkfs.bfs
-rwxr-xr-x 1 root root 43K Feb 13 10:48 /sbin/mkfs.cramfs
-rwxr-xr-x 1 root root 51K Oct 28 15:48 /sbin/mkfs.exfat
lrwxrwxrwx 1 root root 6 Feb 2 07:38 /sbin/mkfs.ext2 -> mke2fs
lrwxrwxrwx 1 root root 6 Feb 2 07:38 /sbin/mkfs.ext3 -> mke2fs
lrwxrwxrwx 1 root root 6 Feb 2 07:38 /sbin/mkfs.ext4 -> mke2fs
-rwxr-xr-x 1 root root 63K Feb 8 2021 /sbin/mkfs.fat
-rwxr-xr-x 1 root root 111K Feb 13 10:48 /sbin/mkfs.minix
lrwxrwxrwx 1 root root 8 Feb 8 2021 /sbin/mkfs.msdos -> mkfs.fat
lrwxrwxrwx 1 root root 6 Oct 31 16:14 /sbin/mkfs.ntfs -> mkntfs
lrwxrwxrwx 1 root root 8 Feb 8 2021 /sbin/mkfs.vfat -> mkfs.fat
```

Вот как с помощью `mkfs` отформатировать целевой диск под файловую систему `ext4`:

1. Сначала запустите `fdisk`, чтобы выбрать нужный диск:
`sudo fdisk -l`
2. Затем внимательно просмотрите вывод и правильно выберите имя диска, который нужно отформатировать (в нашем случае `/dev/sda`).
3. Убедившись, что вы выбрали нужный диск, запустите `mkfs`, чтобы отформатировать его под файловую систему `ext4`. Результат выглядит так:

¹ Здесь `if` — сокр. input file (входной файл), а `of` — сокр. output file (выходной файл). — *Примеч. пер.*

² Сокр. make filesystem (создать файловую систему). — *Примеч. пер.*

```
alexandru@debian:~$ sudo mkfs.ext4 /dev/sda
mke2fs 1.46.6 (1-Feb-2023)
Found a dos partition table in /dev/sda
# Найдена таблица разделов типа dos в /dev/sda
Proceed anyway? (y,N) y      # Все равно продолжить?
Creating filesystem with 1966080 4k blocks and 491520 inodes
# Создается файловая система: 1966080 блоков по 4 Кбайт,
# 491520 индексных дескрипторов
Filesystem UUID: 9dfc7fe4-e4d4-4223-92d6-94dd7e8e6e69
# Идентификатор файловой системы
Superblock backups stored on blocks:
32768, 98304, 163840, 229376, 294912, 819200, 884736, 1605632
# Резервные копии суперблоков сохранены в блоках: ...

Allocating group tables: done
# Выделяются групповые таблицы
Writing inode tables: done
# Записываются таблицы индексных дескрипторов
Creating journal (16384 blocks): done
# Создается журнал (16384 блока)
Writing superblocks and filesystem accounting information: done
# Записываются суперблоки и учетные данные файловой системы
```

У `mkfs` есть несколько ключей. Чтобы отформатировать раздел `ext4`, можно использовать команду `mkfs.ext4`, как мы только что сделали, а можно команду `mkfs` с ключом `-t`, за которым следует тип файловой системы. Ключ `-v` выводит данные в расширенном формате, а ключ `-c` проверяет диск на наличие неисправных секторов, прежде чем создавать файловую систему. Чтобы добавить метку раздела прямо из командной строки, можно использовать ключ `-L`. В следующем примере мы создаем раздел файловой системы `ext4` с именем `newpartition`:

```
sudo mkfs -t ext4 -v -c -L newpartition /dev/sda
```

После того как раздел отформатирован, рекомендуется проверить его на наличие ошибок. Для этого служит утилита `fsck`, которая иногда запускается автоматически после аварийного выключения или через заданные промежутки времени. Как и `mkfs`, она объединяет конкретные программы для наиболее популярных файловых систем. В следующем примере мы запускаем `fsck` на одном из наших разделов и убеждаемся, что проверка не нашла ошибок (об этом свидетельствует слово `clean`):

```
alexandru@debian:~$ sudo fsck -t ext4 /dev/sda
fsck from util-linux 2.38.1
e2fsck 1.46.6 (1-Feb -2023)
/dev/sda: clean, 11/491520 files, 55879/1966080 blocks
```

После того как вы создали раздел, его нужно смонтировать, иначе его нельзя будет использовать.

Монтирование — важная операция в Linux, да и в любой другой операционной системе. Когда вы монтируете диск, вы предоставляете системе доступ к его ресурсам таким образом, чтобы система могла обращаться к нему как к локальному диску. Если в Linux монтируется внешний диск, он связывается с точкой монтирования, которая представляет собой каталог в локальной файловой системе. Точки монтирования — важный компонент операционных систем, совместимых с POSIX, таких как Linux. Смонтированный диск становится доступным для всей операционной системы через точки монтирования. Дополнительные сведения о монтировании см. на странице docs.oracle.com/cd/E19455-01/805-7228/6j6q7ueup/.

Каждый раздел монтируется в пределах существующей структуры файловой системы, причем точка монтирования может размещаться в любом месте иерархии каталогов. Как правило, разные файловые системы монтируются в разные каталоги, которые создаются специально для этого. В следующем разделе мы поговорим о том, как монтировать и демонтировать разделы.

Как монтировать и демонтировать разделы

Утилита **монтирования** в Linux называется `mount`, а **демонтирования** — `umount`. Чтобы узнать, смонтирован ли определенный раздел, можно просто запустить `mount` и посмотреть, что выведет эта команда. Поскольку вывод может оказаться весьма объемным, его можно отфильтровать с помощью `grep`, например:

```
mount | grep /dev/sda
```

В нашем случае `/dev/sda` не отображается в выводе — это значит, что раздел не смонтирован.

Чтобы смонтировать раздел, нужно создать новую точку монтирования. Вот как это можно сделать:

1. Создайте новый каталог, который станет точкой монтирования раздела, — например, каталог USB внутри домашнего каталога:

```
mkdir USB
```

2. Смонтируйте раздел с помощью такой команды:

```
sudo mount /dev/sda ~/USB
```

3. Убедитесь, что к новому разделу теперь можно обратиться. Для примера скопируем на смонтированный накопитель USB файл `mbr-backup`, который мы создали ранее:

```
sudo cp mbr-backup USB/
```

После этого можно проверить, что в каталоге USB появился скопированный файл:

```
alexandru@debian:~$ ls -l USB/
total 4
-rw-r--r-- 1 root root 512 Mar 16 15:44 mbr-backup
```

Команду `mount` нужно запускать с правами суперпользователя. Если вы попытаетесь смонтировать внешнее устройство USB без `sudo`, то получите сообщение об ошибке:

```
alexandru@debian:~$ mount /dev/sda ~/USB
mount: /home/alexandru/USB: must be superuser to use mount.
       dmesg(1) may have more information after failed mount system call.1
```

У `mount` есть много ключей; чтобы увидеть их все, можно вызвать `mount --help`. После того как раздел смонтирован, его можно использовать. А чтобы демонтировать раздел, запустите `umount` — например, так:

```
sudo umount /dev/sda
```

Когда вы демонтируете файловую систему, можете получить сообщения об ошибках, если соответствующий раздел все еще используется — то есть если программы из него загружены в оперативную память и выполняются или если другие активные процессы обращаются к файлам из этого раздела. Поэтому сначала нужно завершить все программы, запущенные из него, а также другие процессы, которые используют эту файловую систему. Иногда не сразу понятно, что именно блокирует файловую систему, и, чтобы разобраться, можно запустить команду `lsof`, которая выведет список всех открытых файлов:

```
sudo lsof | grep /dev/sda
```

Смонтированная файловая система остается доступной только до тех пор, пока компьютер не будет выключен или перезагружен. Если вы хотите смонтировать раздел на постоянной основе, вам понадобится внести соответствующие изменения в файл `/etc/fstab`. Сначала откройте его в вашем любимом текстовом редакторе, например:

```
sudo nano /etc/fstab
```

Затем добавьте в файл новую строку по образцу такой:

```
/dev/sda /mnt/sdb ext4 defaults 0 0
```

Файл `/etc/fstab` содержит конфигурацию для таблицы файловых систем — то есть набор правил, которые определяют, как используются файловые системы. Благодаря этим правилам вам не приходится вручную монтировать и демонтировать каждый диск, и связанные с этим ошибки возникают существенно реже. Таблица состоит из шести столбцов, каждый из которых соответствует определенному параметру. Параметры должны идти в таком порядке:

- **имя устройства.** UUID или имя смонтированного устройства;

¹ Нужно быть суперпользователем, чтобы использовать команду `mount` для каталога `/home/alexandru/USB`. На справочной странице `dmesg(1)` может быть дополнительная информация о неудачном обращении к системному вызову `mount`. — *Примеч. пер.*

- **точка монтирования.** Каталог, в который смонтировано или будет смонтировано устройство;
- **тип файловой системы;**
- **параметры.** Различные параметры, разделенные запятыми;
- **флаг резервного копирования¹;**
- **порядок проверки файловой системы.** Это последняя цифра в строке, которая отвечает за то, в каком порядке программа `fsck` проверяет разделы при запуске: 0 — не проверять, 1 — проверять корневую файловую систему, 2 — проверять другие разделы.

Если данные о монтировании занесены в файл `/etc/fstab`, устройство монтируется в постоянном режиме и подключается снова, когда система инициализируется после того, как ее заново включили или перезагрузили. Обычно файл `/etc/fstab` содержит записи только о разделах и файловых системах внутренних накопителей, а внешние диски или накопители USB автоматически монтируются в каталог `/media` с помощью слоя HAL, который реализован на уровне ядра.

К этому моменту вы уже должны представлять себе, как управлять разделами в Linux, но есть еще один тип разделов, о котором мы не говорили, — **раздел подкачки**. В следующем разделе мы узнаем, как он работает.

Раздел подкачки

Подкачка в Linux реализована весьма надежно. Когда физическая оперативная память исчерпывается, данные перемещаются в виртуальную память, под которую выделяется место на диске. Это дополнительное пространство доступно либо для программ, которые потребляют гораздо меньше памяти, чем им выделено, либо при высокой нагрузке на память. Для подкачки обычно используется один или несколько выделенных разделов, поскольку в Linux может быть несколько областей подкачки. Рекомендуется, чтобы суммарная емкость разделов подкачки была не меньше, чем общий объем оперативной памяти в системе. Чтобы узнать, как фактически используется подкачка, можно просмотреть файл `/proc/swaps` или запустить команду `free`:

```
alexandru@debian:~$ cat /proc/swaps
```

Filename	Type	Size	Used	Priority
/dev/dm-2	partition	999420	300	-2

```
alexandru@debian:~$ free
```

	total	used	free	shared	buff/cache	available
Mem:	48124468	7763840	15675640	69204	25344948	40360628
Swap:	999420	300	999120			

¹ Исторически это поле предназначалось для утилиты резервного копирования `dump`, которая по большей части вытеснена более надежными и современными решениями. — *Примеч. ред.*

Если в вашей системе не настроена подкачка, под нее можно отформатировать специальный раздел и активировать его с помощью таких команд:

```
mkswap /dev/sda1  
swapon /dev/sda1
```

Операционная система кэширует содержимое файлов в памяти, чтобы задействовать подкачку как можно меньше, потому что память работает гораздо быстрее, чем жесткие диски или другие накопители. Подкачка будет использоваться, только если доступной памяти не хватает. Однако в файлы подкачки из памяти выгружаются только те данные, которые размещены в пользовательском пространстве; пространство ядра никогда не выгружается, чтобы не нарушить целостность своих данных. Чтобы пронаблюдать, как используется память в Linux, обратитесь к утилитам, которые мы применяли в главе 5.

Файловые системы и разделы — это главные составляющие любой задачи управления дисками, но время от времени администратор может столкнуться с затруднениями, которые можно решить с помощью новой сущности — логических томов. В следующем разделе мы познакомим вас с ними.

Как управлять логическими томами в Linux

Возможно, вы уже слышали о логических томах, а если нет — сейчас мы кратко расскажем, что это такое. Представьте себе, что на вашем диске закончилось место. В этом случае можно перенести данные на новый диск большей емкости и подключить его вместо старого, но для этого понадобится перезагружать систему и сталкиваться с нежелательными простоями. Вместо этого можно использовать логические тома, которые решают проблему более гибко и эффективно. Они позволяют на лету добавлять новые физические диски в существующие группы томов. При этом данные по-прежнему можно переносить на новый диск, но без простоев: все операции выполняются в рабочей файловой системе.

В Linux логическими томами можно управлять с помощью утилит `pvccreate`, `vgcreate`, `vgdisplay`, `lvcreate`, `lvextend` и `lvdisplay`. Давайте их рассмотрим.

Поскольку в нашей системе пока нет логических томов, мы продемонстрируем, как их создать, на примере другой системы с двумя внутренними дисками: на одном из них установлена операционная система, а второй свободен. Мы будем работать с Debian GNU/Linux, но те же самые команды применимы и в любой другой операционной системе семейства Linux.

Чтобы создать логический том, выполните такие действия:

1. С помощью команды `fdisk` или `lsblk` проверьте, как называются доступные диски:

```
sudo fdisk -l
```

В нашем случае второй, свободный диск — это `/dev/sda`.

2. С помощью команды `pvccreate` инициализируйте физический том, чтобы он поддерживал логические тома:

```
sudo pvccreate /dev/sda
```

Эта команда может не сработать с первого раза и выдать сообщение об ошибке состояния диска, на котором вы хотите инициализировать том. В нашем случае так произошло с одним из дисков, поэтому с него пришлось стереть информацию о файловой системе с помощью утилиты `wipefs`:

```
alexandru@debian:~$ sudo pvccreate /dev/sda
Cannot use /dev/sda: device is partitioned
# Не удастся использовать /dev/sda: устройство сегментировано

alexandru@debian:~$ sudo wipefs --all /dev/sda
/dev/sda: 8 bytes were erased at offset 0x00000200 (gpt): 45 46 49 20 50 41
52 54
/dev/sda: 8 bytes were erased at offset 0x1bf2975e00 (gpt): 45 46 49 20 50 41
52 54
/dev/sda: 2 bytes were erased at offset 0x000001fe (PMBR): 55 aa
/dev/sda: calling ioctl to re-read partition table: Success
```

```
alexandru@debian:~$ sudo pvccreate /dev/sda
Physical volume "/dev/sda" successfully created.
# Физический том "/dev/sda" успешно создан
```

3. С помощью команды `vgcreate` создайте новую группу томов, в которую будет добавлен новый физический том:

```
alexandru@debian:~$ sudo vgcreate newvolume /dev/sda
Volume group "newvolume" successfully created
# Группа томов newvolume успешно создана
```

4. Чтобы просмотреть новую группу томов, запустите команду `vgdisplay`:

```
alexandru@debian:~$ sudo vgdisplay newvolume
\\ --- Volume group ---
  VG Name                newvolume
  System ID
  Format                  lvm2
```

5. Теперь давайте запустим команду `lvcreate`, чтобы создать логический том, используя часть пространства, которое доступно в группе томов. Укажите ключ `-n`, чтобы задать имя логического тома, и ключ `-L`, чтобы задать размер в удобочитаемом виде (мы создали логический том размером 5 Гбайт с именем `projects`):

```
alexandru@debian:~$ sudo lvcreate -n projects -L 5G newvolume
WARNING: ext4 signature detected on /dev/newvolume/projects at offset 1080.
Wipe it? [y/n]: y
# Внимание: обнаружена сигнатура ext4; стереть ее?
Wiping ext4 signature on /dev/newvolume/projects.
Logical volume "projects" created.
```

6. Проверьте, что новый логический том существует:

```
alexandru@debian:~$ sudo ls /dev/mapper/newvolume-projects
/dev/mapper/newvolume-projects
```

7. Чтобы использовать новое устройство, его нужно отформатировать под известную файловую систему и смонтировать так же, как обычный раздел. Сначала отформатируем новый том:

```
alexandru@debian:~$ sudo mkfs -t ext4 /dev/mapper/newvolume-projects
```

8. Теперь можно смонтировать логический том. Сначала создайте новый каталог и смонтируйте в нем том, а затем проверьте его размер с помощью `df`:

```
alexandru@debian:~$ mkdir LVM
alexandru@debian:~$ sudo mount /dev/mapper/newvolume-projects ~/LVM

alexandru@debian:~$ df -h ~/LVM/
Filesystem                Size      Used Avail Use% Mounted on
/dev/mapper/newvolume-projects  4.9G    24K    4.6G   1% /home/alexandru/LVM
```

9. Все изменения, которые мы внесли до сих пор, не сохранятся после перезагрузки компьютера. Чтобы зафиксировать их, нужно отредактировать файл `/etc/fstab`, добавив в него такую строку:

```
/dev/mapper/newvolume-projects /home/alexandru/LVM ext4 defaults 1 2
```

10. Теперь можно проверить свободное место на логическом томе и расширить его, если нужно. Чтобы вывести сведения о томе, запустите `vgdisplay`:

```
sudo vgdisplay newvolume
```

11. Теперь можно расширить логический том с помощью команды `lvextend`. Давайте добавим к нему 5 Гбайт, чтобы в итоге получить том емкостью 10 Гбайт:

```
alexandru@debian:~$ sudo lvextend -L +5G /dev/mapper/newvolume-projects
Size of logical volume newvolume/projects changed from 5.00 GiB (1280 extents) to 10.00 GiB (2560 extents).
Logical volume newvolume/projects successfully resized.
```

12. Наконец, давайте с помощью `resize2fs` изменим размер файловой системы так, чтобы она соответствовала новой емкости логического тома. После этого ее размер можно проверить с помощью `df`:

```
alexandru@debian:~$ sudo resize2fs -p /dev/mapper/newvolume-projects
resize2fs 1.46.6 (1-Feb-2023)
Filesystem at /dev/mapper/newvolume-projects is mounted on /home/alexandru/LVM; on-line resizing required
old_desc_blocks = 1, new_desc_blocks = 2
The filesystem on /dev/mapper/newvolume-projects is now 2621440 (4k) blocks long.
```

Логические тома — это продвинутая тема, которую необходимо освоить любому системному администратору Linux. Краткие практические примеры, которые

мы привели в этом разделе, демонстрируют только основные приемы работы с логическими томами. Не бойтесь глубже погрузиться в эту тему, если понадобится.

В следующем разделе пойдет речь о том, как создавать полные снимки файловой системы на логическом томе.

Снимки логических томов

Что такое снимок логического тома? Это «застывший» экземпляр тома. Более конкретно: в нем используется технология **копирования при записи (COW, сору-оп-write)**, которая отслеживает каждый блок исходного тома, и каждый раз, когда тот или иной блок меняется в результате операций записи, его значение копируется в снимок.

Снимки создаются постоянно и мгновенно и сохраняются до тех пор, пока их не удалят. В результате из любого снимка можно создать резервную копию. Копирование при записи приводит к тому, что снимки постоянно меняются, поэтому, прежде чем их создавать, имеет смысл продумать, сколько места им потребуется. По возможности учитывайте, какой объем данных может измениться, пока существует снимок. Как только снимок заполнится, он автоматически отключится.

Как создать новый снимок логического тома

Чтобы создать новый снимок, запустите команду `lvcreate` с ключом `-s`. При этом также можно указать размер снимка с помощью ключа `-L` и добавить его имя с помощью ключа `-n`:

```
alexandru@debian:~$ sudo lvcreate -s -L 5G -n linux-snapshot-01 /dev/mapper/
newvolume-projects
Logical volume "linux-snapshot-01" created.
```

В предыдущей команде мы задали размер снимка 5 Гбайт и назвали его `linux-snapshot-01`. Последний аргумент команды — местоположение тома, для которого создается снимок. Чтобы вывести сведения о новом снимке, используйте команду `lvs`:

```
alexandru@debian:~$ sudo lvs
LV          VG          Attr      LSize Pool Origin   Data% Meta% Move Log Cpy%Sync Convert
linux-snapshot-01 newvolume swi-a-s--- 5.00g          projects 0.01
projects    newvolume owi-aos--- 10.00g
```

Чтобы получить больше сведений о логических томах, запустите `lvdisplay`. Вы увидите информацию обо всех томах, среди которых будет и снимок, который мы только что создали.

Когда мы создавали снимок, мы задали ему размер 5 Гбайт. Теперь давайте с помощью команды `lvextend` расширим его до размера исходного тома (10 Гбайт):

```
alexandru@debian:~$ sudo lvextend -L +5G /dev/mapper/newvolume-linux--
snapshot--01
Size of logical volume newvolume/linux-snapshot-01 changed from 5.00 GiB
(1280 extents) to 10.00 GiB (2560 extents).
Logical volume newvolume/linux-snapshot-01 successfully resized.
```

Как видите, имя снимка, которое передается в последнем аргументе, отличается от того имени, которое мы ему присвоили. Мы назвали том моментального снимка `linux-snapshot-01`, но если заглянуть в каталог `/dev/mapper/`, то мы увидим, что в соответствующее имя добавились еще два дефиса. Это соглашение, в соответствии с которым принято представлять файлы логических томов.

После того как вы научились создавать снимки логических томов, давайте узнаем, как восстановить логический том из снимка.

Как восстановить логический том из снимка

Чтобы восстановить логический том, сначала нужно демонтировать его файловую систему. Для этого воспользуемся командой `umount`:

```
sudo umount ~/LVM
```

Затем можно восстановить том с помощью команды `lvconvert`. После того как снимок будет объединен с источником, результат можно проверить командой `lvs`:

```
alexandru@debian:~$ sudo lvconvert --merge /dev/mapper/newvolume-linux--snapshot--01
Merging of volume newvolume/linux-snapshot-01 started.
newvolume/projects: Merged: 100.00%

alexandru@debian:~$ sudo lvs
LV          VG          Attr   LSize Pool Origin Data% Meta% Move Log Cpy%Sync Convert
projects newvolume -wi-a----- 10.00g
```

После объединения снимок автоматически удаляется.

Теперь мы рассмотрели основы работы с логическими томами в Linux: эта технология сложнее, чем обычное сегментирование диска. Возможно, кого-то это отпугнет, но ее сильные стороны нередко могут пригодиться на практике. Впрочем, у логических томов есть и недостатки: например, они способны добавить лишних хлопот в случае аварийного восстановления или при аппаратном сбое. И все же знать об этой технологии определенно не помешает.

Итоги

Любому системному администратору Linux часто требуется управлять файловыми системами и дисками, поэтому важно уметь работать с устройствами в Linux, а также форматировать и сегментировать диски. Кроме того, стоит знать о логических томах, потому что они обеспечивают гибкий способ управления разделами.

Эти навыки дадут вам прочную основу для того, чтобы справляться с любыми типичными задачами администрирования. В следующей главе вы познакомитесь с обширной областью **сетевого окружения** в Linux.

Вопросы

В завершение этой главы стоит вспомнить несколько важных подробностей о файловой системе и управлении дисками в Linux:

1. Вспомните еще один инструмент для работы с дисками и установите его.

Подсказка: попробуйте установить `parted` и запустить его из командной строки, а также обратите внимание на графический интерфейс GParted.

2. Поэкспериментируйте с приложением «Диски» (Disks) в GNOME и Диспетчером разделов от KDE (KDE Partition Manager) и сравните этот опыт с работой в командной строке.

Подсказка: откройте каждое из приложений на экране бок о бок с терминалом. Попробуйте отформатировать и смонтировать диск в командной строке и посмотрите, как это отразится в графическом интерфейсе.

3. Попробуйте форматировать новые разделы под различные файловые системы.

Подсказка: используйте `btrfs` вместо `ext4`.

4. Подробно изучите, как устроены ваши диски и файловые системы.

Подсказка: используйте такие инструменты, как `lsblk`, `df` и `fdisk`.

7

Сетевое окружение Linux

Сетевое окружение Linux — это обширнейшая область. За последние десятилетия написано бесчисленное количество монографий и справочников, посвященных тонкостям сетевого администрирования в Linux. Иногда не только новичкам, но и опытным пользователям приходится прикладывать немало усилий, чтобы просто усвоить основные понятия. В этой главе представлен относительно краткий обзор сетевых технологий Linux с упором на уровни сетевого взаимодействия, сокеты и порты, сетевые службы и протоколы, а также сетевую безопасность.

Мы надеемся, что эта глава послужит удобным введением в основные принципы работы с сетями Linux для начинающих пользователей, а профессиональным администраторам она поможет освежить свои знания.

В этой главе мы рассмотрим такие темы:

- Основы сетевых технологий: как устроены компьютерные сети, сетевые модели, протоколы, сетевые адреса и порты. Мы также рассмотрим, как на практике настраивать некоторые сетевые параметры Linux с помощью командной строки.
- Как работать с сетевыми службами: вы познакомитесь с распространенными сетевыми серверами, которые применяются в Linux.
- Как обеспечивать безопасность сетевого окружения.

Технические требования

На протяжении всей этой главы мы будем в той или иной степени использовать командную строку Linux. Чтобы выполнять упражнения, вам понадобится рабочий дистрибутив Linux, который установлен либо в виртуальной машине, либо на настольном компьютере. Если у вас его еще нет, вернитесь к главе 1 «Как установить Linux», где описывается процесс установки. Большинство команд и примеров в этой главе демонстрируются в Ubuntu и Fedora, но все то же самое применимо и к любой другой платформе Linux.

Основы сетевых технологий

Сегодня практически невозможно представить себе компьютер, который не подключен к какой-нибудь сети или интернету. Мы постоянно наращиваем свое

присутствие в сети, пользуемся облачными вычислениями, мобильной связью и интернетом вещей: все это было бы невозможно без высокораспределенных, высокоскоростных и масштабируемых сетей, которые обслуживают соответствующий трафик. Однако основные принципы сетевых технологий, на которые опирается современный интернет, насчитывают уже несколько десятилетий. Все эти технологии продолжают развиваться, но некоторые исходные примитивы и понятия еще долго будут влиять на то, как формируются компоненты будущих коммуникаций.

В этом разделе вы познакомитесь с некоторыми из этих основных принципов и, надеемся, почувствуете желание изучать эту тему глубже. Начнем с компьютерных сетей.

Компьютерные сети

Компьютерная сеть — это группа из двух или более компьютеров (или узлов), которые соединены физической средой (кабельной, беспроводной, оптической) и взаимодействуют друг с другом с помощью стандартного набора согласованных протоколов связи. На самом высоком уровне **инфраструктура сетевых коммуникаций** состоит из компьютеров, вспомогательных устройств, коммутаторов, маршрутизаторов, патчкордов или оптических кабелей, беспроводных сред и всевозможного сетевого оборудования.

Помимо физического подключения и расположения, сети также характеризуются логической схемой, которая строится с помощью сетевых топологий, уровней инфраструктуры и соответствующих потоков данных. Характерный пример логической сетевой иерархии — трехуровневая структура, которая состоит из **демилитаризованной зоны (DMZ)**, **межсетевого экрана** и **локальных сетей**. DMZ — это внешний уровень сетевого окружения с дополнительной защитой от угроз из общедоступного интернета. Сетевой экран контролирует трафик между DMZ и локальной сетью.

Сетевые устройства можно идентифицировать по двум параметрам:

- **Сетевые адреса** помогают определять местоположение узлов в сети с помощью **протоколов связи**, таких как **IP** (подробнее см. «Протоколы TCP/IP» далее в этой главе).
- **Имена узлов**. Это названия устройств в человекочитаемом формате, которые легче запомнить, чем сетевые адреса.

Компьютерные сети принято классифицировать по масштабу и охвату. По этим признакам выделяются **локальные** и **глобальные сети**:

- **Локальные сети (LAN)**. Локальная сеть — это группа устройств, которые соединены друг с другом и физически расположены в одном месте — например, в частном доме, школе или офисе. Локальные сети бывают разные — от домашней сети с несколькими устройствами до крупных корпоративных сетей с тысячами пользователей и компьютеров.

Независимо от размера, основной признак локальной сети состоит в том, что она объединяет устройства на одной ограниченной территории. Примерами таких сетей могут служить домашняя сеть или бесплатный Wi-Fi в местной кофейне.

Подробнее узнать о локальных сетях можно на странице cisco.com/c/en/us/products/switches/what-is-a-lan-local-area-network.html.

Когда компьютерная сеть охватывает несколько регионов или несколько взаимосвязанных локальных сетей, она называется глобальной.

- **Глобальные сети (WAN).** Глобальная сеть — это, как правило, сеть сетей, в которой несколько локальных сетей обмениваются данными друг с другом. В этом смысле интернет можно рассматривать как крупнейшую в мире глобальную сеть. Другой характерный пример глобальной сети — компьютерная сеть международной компании с многочисленными офисами по всему миру. Некоторые глобальные сети принадлежат провайдерам, которые предоставляют услуги доступа различным предприятиям и учреждениям.

Глобальные сети делятся на несколько категорий, в зависимости от их структуры, охвата и назначения. Типичные примеры глобальных сетей — **персональные сети (PAN)**, **городские сети (MAN)** и **облачные сети (IAN)**.

Подробнее узнать о глобальных сетях можно на странице cisco.com/c/en/us/products/switches/what-is-a-wan-wide-area-network.html.

Чтобы познакомиться с фундаментальными принципами работы компьютерных сетей, нам с вами понадобится затронуть теоретическую модель, которая описывает сетевые коммуникации в целом. Сейчас мы ее рассмотрим.

Модель OSI

Модель **OSI**¹ — это теоретическое представление многоуровневого механизма связи между компьютерными системами, которые взаимодействуют друг с другом через сеть. Эту модель ввела в 1983 году **Международная организация по стандартизации (ISO)** в качестве стандарта, который позволяет различным компьютерным системам обмениваться данными друг с другом.

Модель OSI можно рассматривать как универсальную схему сетевых коммуникаций². Как показано на следующем рисунке, она определяет стек из семи уровней, через которые проходит поток данных.

¹ Сокр. Open Systems Interconnection (взаимодействие открытых систем). — *Примеч. пер.*

² В профессиональном сообществе существуют разные мнения о том, насколько модель OSI полезна на практике. Некоторые специалисты считают, что она с самого начала была слишком абстрактной и нежизнеспособной и с тех пор окончательно отделилась от современных реалий. Другие полагают, что эта модель актуальна как эталонная теоретическая схема, которая обобщает большинство существующих реализаций. Многие эксперты придерживаются промежуточных или альтернативных позиций. — *Примеч. ред.*

Уровень		Тип данных	Описание
7	Прикладной	Данные	Взаимодействие с пользователем и API высокого уровня
6	Представления		Перевод данных в пригодный для использования формат (например, шифрование/дешифрование, кодирование/декодирование, сжатие/декомпрессия)
5	Сеансовый		Сеансы связи (соединения, сокет, порты)
4	Транспортный	Сегменты, дейтаграммы	Надежная передача сегментов данных с помощью протоколов управления передачей (например, TCP или UDP)
3	Сетевой	Пакеты	Пакеты данных с информацией об адресации, маршрутизации и управлении трафиком
2	Канальный	Кадры	Упаковывание данных в виде устойчивых фреймов
1	Физический	Биты	Передача и прием необработанных потоков битов по физическому носителю

Рис. 7.1. Модель OSI

В многоуровневом представлении, которое показано на предыдущем рисунке, на передающем конце коммуникационный поток движется сверху вниз, а на принимающем — снизу вверх. Прежде чем мы подробнее рассмотрим каждый уровень, позвольте вкратце объяснить, как устроена инкапсуляция и декапсуляция данных в модели OSI.

Инкапсуляция и декапсуляция данных в модели OSI

Передача данных по сети с одного компьютера на другой регулируется определенными правилами. В модели OSI потоки данных могут следовать в двух направлениях. Одно направление — сверху вниз (от 7-го уровня до 1-го), когда данные **инкапсулируются**, а другое — снизу вверх (от 1-го уровня до 7-го), когда они **декапсулируются**. Инкапсуляция происходит, когда один компьютер передает данные другому, а декапсуляция — когда компьютер их получает. Давайте рассмотрим эти операции подробнее:

- **Инкапсуляция.** Когда один компьютер передает данные другому, они проходят через все уровни стека, где преобразуются и обрастают дополнительной информацией. На прикладном уровне (7) пользователь непосредственно взаимодействует с приложением. Затем данные проходят через уровень представления (6) и сеансовый уровень (5), где преобразуются в пригодный для использования формат. На транспортном уровне (4) данные разбиваются на более мелкие фрагменты (сегменты) и снабжаются новыми заголовками TCP. На сетевом уровне (3) данные превращаются в пакеты, получают

заголовки IP и отправляются на канальный уровень (2), где они рассматриваются как кадры и содержат заголовки как TCP, так и IP. На уровне 2 каждый кадр дополняется аппаратными адресами (MAC-адресами) отправителя и получателя, а также заголовками протокола **LLC**. В этот момент добавляется новое поле **FCS**, которое позволяет проверять ошибки. Затем кадры проходят через физический уровень (1).

- **Декапсуляция.** Когда компьютер получает данные, они следуют по тому же маршруту, но в противоположную сторону. Путь начинается с физического уровня (1), где происходит первая синхронизация, после чего принятый кадр отправляется на канальный уровень (2), где проверяется на ошибки с помощью поля FCS — это происходит по алгоритму **CRC**. После этого данные уже являются пакетом и отправляются через все остальные уровни. Здесь заголовки, которые добавлялись в процессе инкапсуляции, удаляются, пока данные не достигнут верхнего уровня, на котором их сможет использовать целевой компьютер. Графически это можно представить так:



Рис. 7.2. Инкапсуляция и декапсуляция в модели OSI

Давайте подробно рассмотрим каждый из этих уровней и узнаем, как он участвует в сетевом взаимодействии.

Физический уровень

Физический уровень (уровень 1) состоит из сетевого оборудования и инфраструктуры, которая соединяет устройства и обслуживает связь, — например, сюда входят кабели, беспроводные или оптические среды, разъемы и коммутаторы. Этот уровень преобразовывает потоки битов в сигналы среды передачи (например, электрические, оптические или радиосигналы) и обратно, регулируя при этом скорость передачи данных.

На физическом уровне работают такие протоколы, как Ethernet, **USB** и **DSL**.

Канальный уровень

Канальный уровень (уровень 2) регулирует устойчивый поток данных между двумя устройствами, которые подключены напрямую, будь то сопряженные

узлы в глобальной сети или компьютеры в локальной сети. Одна из задач канального уровня — управлять потоком данных, адаптируясь к скорости передачи данных на физическом уровне. На принимающем устройстве канальный уровень исправляет ошибки связи, которые возникли на физическом уровне. Канальный уровень состоит из двух подсистем:

- **MAC** (Media Access Control, управление доступом к среде) использует MAC-адреса, чтобы идентифицировать и соединять друг с другом устройства в сети. Эта подсистема также контролирует права доступа устройств к передаче и получению данных по сети.
- **LLC** (Logical Link Control, управление логическим соединением) распознает и инкапсулирует протоколы сетевого уровня, а также проверяет ошибки и синхронизирует кадры при передаче или получении данных.

Стандартные фрагменты данных, которыми управляет канальный уровень, также известны как **кадры**. Кадр — это единица передачи данных, которая служит контейнером для одного сетевого пакета. Сетевые пакеты обрабатываются на следующем уровне OSI (сетевом). Когда несколько устройств одновременно обращаются к одному и тому же физическому уровню, могут возникать конфликты кадров. Протоколы канального уровня умеют не только обнаруживать и разрешать, но и предотвращать их.

Существуют также кадры Ethernet, которые представляют собой инкапсулированные данные, определенные для реализаций MAC. Кроме того, кадры бывают в оригинальном формате IEEE 802.3 Ethernet или в таких форматах, как IEEE 802.2 SNAP или Ethernet II.

Еще один пример канального протокола — **PPP**, двоичный сетевой протокол, который используется в высокоскоростных широкополосных сетях связи.

Сетевой уровень

Сетевой уровень (уровень 3) определяет оптимальный маршрут, по которому передаются данные между сетевыми устройствами. Чтобы перемещать пакеты данных от отправителя к получателю, используется механизм маршрутизации, который основан на IP-адресах устройств, участвующих в обмене данными.

На передающем конце сетевой уровень принимает данные от транспортного уровня и разбирает их на сетевые пакеты. На принимающем конце этот уровень принимает кадры от нижележащего канального уровня и формирует из них пакеты.

На сетевом уровне работает протокол ICMP, с помощью которого сетевые устройства диагностируют проблемы сетевого взаимодействия. ICMP сигнализирует об ошибке, когда запрошенная оконечная точка недоступна, отправляя сообщения типа «сеть назначения недостижима», «истекло время ожидания», «маршрут источника не функционирует» и др.

Транспортный уровень

Транспортный уровень (уровень 4) работает с **сегментами данных**, или **дейтаграммами**. Он в основном отвечает за передачу данных от отправителя к получателю и гарантирует определенное **качество обслуживания (QoS)**. На передающем конце данные, которые поступили с вышележащего уровня (сеансового), разбираются на сегменты. На принимающем конце транспортный уровень вновь собирает в пакеты данных сегменты, полученные с нижележащего уровня (сетевого).

Транспортный уровень поддерживает надежность передачи данных с помощью механизмов управления потоком и контроля ошибок. Управление потоком регулирует скорость передачи данных между оконечными точками с разной скоростью соединения, чтобы отправитель не перегружал получателя. Если полученные данные неверны, функция контроля ошибок может запросить повторную передачу данных.

На транспортном уровне работают такие протоколы, как TCP и UDP.

Сеансовый уровень

Сеансовый уровень (уровень 5) управляет жизненным циклом каналов связи (или сеансов) между устройствами, которые обмениваются данными по сети. На этом уровне сеансы или сетевые соединения обычно определяются сетевыми адресами, сокетами и портами — мы объясним каждое из этих понятий в разделах «Сокеты и порты» и «IP-адреса». Сеансовый уровень отвечает за целостность передачи данных в рамках канала связи или сеанса. Например, если сеанс прерывается, то передача данных возобновляется с предыдущей контрольной точки.

К типичным протоколам сеансового уровня относятся **RPC**, который служит для межпроцессного взаимодействия, и **NetBIOS** — протокол совместного использования файлов и разрешения имен.

Уровень представления

Уровень представления (уровень 6) позволяет преобразовывать данные между вышележащим прикладным уровнем и нижележащим сеансовым уровнем. На передающем конце данные переводятся в системно-независимую форму перед тем, как будут отправлены по сети. На принимающем конце данные преобразуются в формат, который нужен приложению. Примеры таких преобразований — шифрование и дешифрование, сжатие и декомпрессия, кодирование и декодирование, а также сериализация и десериализация.

Обычно между уровнем представления и прикладным уровнем нет четкой границы, главным образом потому, что различные форматы данных тесно связаны с приложениями, которые их потребляют. К стандартным форматам представления данных относятся ASCII, XML, JSON, JPEG, ZIP и др.

Прикладной уровень

Прикладной уровень (уровень 7) находится ближе всего к конечным пользователям. Этот уровень обрабатывает ввод или вывод данных приложения в той или иной осмысленной форме. Прикладной уровень не содержит самих приложений и не запускает их, а выступает в качестве абстракции между приложениями, представляя компонент связи и нижележащую сеть. Браузеры и почтовые клиенты — типичные примеры приложений, которые взаимодействуют с прикладным уровнем.

К протоколам уровня 7 относятся DNS, HTTP, FTP, а также почтовые протоколы POP, IMAP и SMTP.

Прежде чем закончить этот раздел, отметим, что модель OSI — это обобщенная схема уровней сетевого взаимодействия и его теоретическая основа. Похожей, но более практичной иллюстрацией сетевого стека служит модель TCP/IP. Обе эти модели полезны для того, чтобы проектировать и реализовывать сетевые компоненты, а также диагностировать и устранять сетевые неполадки. Модель OSI обеспечивает всеобъемлющее представление всего сетевого стека — от физической среды до прикладного уровня, и на каждом уровне фигурируют свои типы блоков данных (PDU) и механизмы передачи данных. Однако модель TCP/IP гораздо проще, в ней несколько уровней модели OSI сводятся в один, а механизмы передачи данных больше ориентируются на протоколы. Мы поговорим об этом подробнее в следующем разделе.

Модель сетевого стека TCP/IP

Модель TCP/IP — это четырехуровневая интерпретация сетевого стека OSI, в которой некоторые смежные уровни OSI объединены, как показано на следующем рисунке:

	Модель OSI	Модель TCP/IP	Протоколы TCP/IP
7	Прикладной уровень	Прикладной уровень	DNS, HTTP, FTP, SMTP, SNMP, Telnet, ...
6	Уровень представления		
5	Сеансовый уровень		
4	Транспортный уровень	Транспортный уровень	TCP, UDP, ...
3	Сетевой уровень	Межсетевой уровень	IP, ARP, ICMP, IGMP, ...
2	Канальный уровень	Канальный уровень	Ethernet, Token Ring, Frame Relay, ATM, ...
1	Физический уровень		

Рис. 7.3. Модели OSI и TCP/IP

Хронологически модель TCP/IP старше, чем OSI. Впервые ее предложило Министерство обороны США в рамках проекта по созданию системы межсетевого взаимодействия, который разрабатывало агентство **DARPA**. Этот проект в итоге превратился в современный интернет.

Уровни модели TCP/IP выполняют те же функции, что и аналогичные уровни OSI. Давайте вкратце рассмотрим каждый из них.

Канальный уровень

Канальный уровень (или **уровень сетевого интерфейса**) обеспечивает передачу данных по физической среде (например, проводной, беспроводной или оптической). На этом уровне работают такие сетевые протоколы, как Ethernet, Token Ring и Frame Relay, а в модели OSI ему соответствуют физический и канальный уровни.

Межсетевой уровень

Межсетевой уровень (или **уровень интернета**) обеспечивает доставку данных от одного узла сети к другому *без установки соединения*. Соответствующие протоколы описывают сетевое взаимодействие, при котором отправитель передает данные получателю, предварительно не устанавливая сеанс связи с ним. Межсетевой уровень отвечает за то, чтобы на передающем конце данные упаковывались в сетевые пакеты, а на принимающем — распаковывались обратно. На этом уровне функционируют механизмы маршрутизации, которые определяют оптимальный путь между узлами сети. Это соответствует сетевому уровню в модели OSI.

Транспортный уровень

Транспортный уровень, как и в модели OSI, поддерживает сеансы связи между соединенными сетевыми узлами. На этом уровне реализуются механизмы, которые обнаруживают и исправляют ошибки, чтобы обеспечить надежную передачу данных между оконечными точками.

Прикладной уровень

Прикладной уровень обеспечивает абстракцию обмена данными между программными приложениями и нижележащей сетью. Этот уровень соответствует совокупности сеансового уровня, уровня представления и прикладного уровня в модели OSI.

Как уже говорилось в этой главе, модель TCP/IP — это представление сетевого стека, ориентированное на протоколы. Эта модель легла в основу всего интернета, и в ее рамках постепенно определялись и развивались сетевые протоколы, которые необходимы для обмена данными в интернете. Эти протоколы в совокупности называются *стеком TCP/IP*. В следующем разделе описаны самые важные из них.

Протоколы TCP/IP

В этом разделе мы перечислим некоторые широко используемые сетевые протоколы. Этот перечень — не всеобъемлющий справочник: в стек TCP/IP входит огромное количество протоколов, и нет возможности подробно рассмотреть их все в рамках этой главы. Однако несколько протоколов стоит изучить, потому что они особенно часто используются в повседневной сетевой коммуникации и административных задачах.

В следующем списке кратко описывается каждый из ключевых протоколов TCP/IP и соответствующий ему идентификатор **RFC**¹. Стандарт RFC — это подробная техническая документация на протокол или на что-то еще; такие документы обычно выпускает **Инженерный совет интернета (IETF)**. Подробнее о RFC можно узнать на странице ietf.org/standards/rfcs. Вот самые важные протоколы:

- **IP** (Internet Protocol, протокол интернета, *RFC 791*) идентифицирует сетевые узлы по адресам фиксированной длины, которые также известны как IP-адреса (мы подробнее поговорим о них в следующем разделе). Протокол IP использует дейтаграммы в качестве единицы передачи данных, а также умеет фрагментировать и дефрагментировать крупные дейтаграммы, чтобы компенсировать недостатки сетей с небольшими пакетами (и избежать задержек при передаче). Протокол IP также предоставляет механизмы маршрутизации, которые находят оптимальный путь передачи данных между узлами сети. IP работает на сетевом уровне (уровень 3) в модели OSI.
- **ARP** (Address Resolution Protocol, протокол разрешения адресов, *RFC 826*) используется протоколом IP, чтобы сопоставлять сетевые адреса **IPv4** с MAC-адресами устройств, с которыми работают канальные протоколы. ARP функционирует на канальном уровне (уровень 2) в модели OSI.
- **NDP** (Neighbor Discovery Protocol, протокол обнаружения соседей, *RFC 4861*) похож на ARP, но сопоставляет адреса **IPv6**. NDP также работает на канальном уровне (уровень 2) в модели OSI.
- **ICMP** (Internet Control Message Protocol, протокол межсетевых управляющих сообщений, *RFC 792*) — это вспомогательный протокол, который обрабатывает ошибки передачи данных. Если устройство или узел недоступен в течение заданного времени ожидания, ICMP сообщает об ошибке. Этот протокол работает на сетевом уровне (уровень 3) в модели OSI.
- **TCP** (Transmission Control Protocol, протокол управления передачей, *RFC 793*) — высоконадежный протокол с установкой соединения. TCP требует, чтобы узлы устанавливали логическое соединение (например, путем *квитирования*) перед тем, как обмениваться данными. TCP работает на транспортном уровне (уровень 4) в модели OSI.

¹ Сокр. Request for Comments (запрос на комментарии). — *Примеч. пер.*

- **UDP** (User Datagram Protocol, протокол пользовательских дейтаграмм, *RFC 768*) — протокол связи без установки соединения. В отличие от TCP, в UDP нет механизма квитирования, а следовательно, он не гарантирует доставку данных. UDP использует дейтаграммы в качестве единиц передачи данных и подходит для сетевых коммуникаций, где можно обойтись без проверки ошибок. Он работает на транспортном уровне (уровень 4) в модели OSI.
- **DHCP** (Dynamic Host Configuration Protocol, протокол динамической настройки узла, *RFC 2131*) предоставляет механизмы для того, чтобы запрашивать и передавать сведения о конфигурации узла, которые необходимы устройствам в сети TCP/IP. DHCP позволяет автоматически (динамически) выделять повторно используемые IP-адреса и другие ресурсы конфигурации. С точки зрения модели OSI протокол DHCP относится к прикладному уровню (уровень 7) в модели OSI, но инициализирующий механизм обнаружения DHCP работает на канальном уровне (уровень 2).
- **DNS** (Domain Name System, система доменных имен, *RFC 6895*) — это протокол, который работает как сетевая адресная книга, где узлы в сети идентифицируются по человекочитаемым именам вместо IP-адресов. Когда сетевое соединение запрашивает доменное имя удаленного устройства (например, `dns.google.com`), DNS преобразует его в IP-адрес (например, `8.8.8.8`). Протокол DNS работает на прикладном уровне (уровень 7) в модели OSI.
- **HTTP** (HyperText Transfer Protocol, протокол передачи гипертекста, *RFC 9110–9112*) — это «общий язык» интернета. HTTP — протокол прикладного уровня, основанный на модели «запрос — ответ» между клиентским приложением (например, браузером) и оконечной точкой сервера (например, веб-сервером). Этот протокол поддерживает широкий спектр форматов данных — от текста до изображений и видеопотоков. HTTP работает на прикладном уровне (уровень 7) в модели OSI.
- **FTP** (File Transfer Protocol, протокол передачи файлов, *RFC 959*) — стандартный протокол для передачи файлов, которые клиент FTP запрашивает с сервера FTP. FTP работает на прикладном уровне (уровень 7) в модели OSI.
- **TELNET** (Telecommunications Network, сеть телекоммуникаций, *RFC 854*) — протокол прикладного уровня, который обеспечивает двунаправленную текстовую связь между клиентом и сервером, используя виртуальное терминальное соединение. TELNET работает на прикладном уровне (уровень 7) в модели OSI.
- **SSH** (Secure Shell, защищенная оболочка, *RFC 4253*) — защищенный протокол прикладного уровня, который реализует стойкое шифрование и криптографическую аутентификацию узлов. SSH использует виртуальное терминальное соединение между клиентом и сервером и работает на прикладном уровне (уровень 7) в модели OSI.

- **SMTP** (Simple Mail Transfer Protocol, простой протокол передачи почты, *RFC 5321*) — протокол прикладного уровня, который позволяет отправлять и получать электронные сообщения между почтовым клиентом (например, Outlook) и почтовым сервером (например, Exchange Server). SMTP поддерживает стойкое шифрование и аутентификацию узлов и действует на прикладном уровне (уровень 7) в модели OSI.
- **SNMP** (Simple Network Management Protocol, простой протокол управления сетью, *RFC 1157*) предназначен для удаленного управления и мониторинга устройств. SNMP работает на прикладном уровне (уровень 7) в модели OSI.
- **NTP** (Network Time Protocol, протокол сетевого времени, *RFC 5905*) обеспечивает синхронизацию системных часов на разных узлах сети. NTP работает на прикладном уровне (уровень 7) в модели OSI.

Большинство протоколов, которые здесь перечислены, идентифицируют устройства, которые обмениваются данными, с помощью протокола IP. Каждое устройство в сети однозначно идентифицируется по IP-адресу. Давайте рассмотрим эти адреса подробнее.

IP-адреса

IP-адрес — это уникальный идентификатор фиксированной длины, который соответствует устройству в сети. Устройства распознают друг друга и взаимодействуют друг с другом по IP-адресам. Система адресации IP похожа на систему уличных почтовых адресов, на которые ориентируются службы доставки.

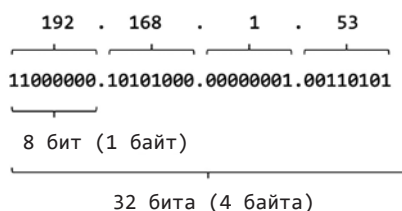
Изначально IP-адрес определялся как 32-битное число — сейчас такие числа называются адресами **IPv4**. С ростом интернета множество доступных IP-адресов было практически исчерпано, и, чтобы решить эту проблему, в новой версии протокола IP была представлена 128-битная схема нумерации адресов. 128-битный IP-адрес также известен как адрес **IPv6**.

В следующих нескольких разделах мы подробнее рассмотрим понятия, которые тесно связаны с IP-адресами, такие как форматы адресов IPv4 и IPv6, классы сетей, подсети и широковещательные адреса.

Адреса IPv4

Адрес IPv4 — это 32-битное (4-байтовое) число, которое обычно выражается в виде четырех групп однобайтовых (8-битных) чисел, разделенных точкой (.). В каждой из групп находится целое число в диапазоне от 0 до 255, например: 192.168.1.53.

На следующем рисунке показано двоичное представление адреса IPv4:

**Рис. 7.4.** Формат адреса IPv4

Адресное пространство IPv4 ограничено 4 294 967 296 адресами (2^{32} , то есть около 4 млрд адресов). Из них примерно 18 млн зарезервированы для специальных целей (например, для частных сетей), а около 270 млн — для многоадресной рассылки.

Адрес многоадресной рассылки — это логический идентификатор группы IP-адресов; подробнее см. *RFC 6308* (tools.ietf.org/html/rfc6308).

Классы сетей

На заре интернета старший байт (первая группа) в адресе IPv4 обозначал **номер сети**, последующие байты определяли иерархию сети и подсети, а самый младший байт идентифицировал оконечное устройство. Этой схемы вскоре стало не хватать, потому что она позволяла эксплуатировать только $2^8 = 256$ сетей, каждой из которых соответствует уникальный старший байт адреса. По мере того как возникали все новые сети, которые нужно было идентифицировать, систему IP-адресов требовалось пересмотреть, чтобы учесть фактическую ситуацию. Эту проблему решила новая спецификация IP, которая была представлена в 1981 году (*RFC 791*) и разделила адресное пространство IPv4 на пять классов на основе старших 4 битов адреса, как показано на следующем рисунке:

Класс	Ведущие биты	Начальный адрес	Конечный адрес	Маска подсети по умолчанию
A	0	0.0.0.0	127.255.255.255	255.0.0.0
B	10	128.0.0.0	191.255.255.255	255.255.0.0
C	110	192.0.0.0	223.255.255.255	255.255.255.0
D (многоадресная рассылка)	1110	224.0.0.0	239.255.255.255	Не определена
E (зарезервировано)	1111	240.0.0.0	255.255.255.255	Не определена

Рис. 7.5. Классы сетей

Дополнительную информацию о классах сети можно найти в *RFC 1700* (datacenter.ietf.org/doc/rfc1700)¹. На предыдущем рисунке в последнем столбце указана маска подсети по умолчанию для каждого из этих классов. Далее мы рассмотрим подсети.

Подсети

Подсети — это логические подразделения сети IP. Подсети были введены для того, чтобы идентифицировать устройства, которые принадлежат к одной сети. Всем устройствам в одной сети соответствует одна и та же старшая группа битов в IP-адресах. Понятие подсети позволяет логически разделить IP-адрес на две части: **адрес сети** (или **префикс сети**) и **адрес узла**. Численное представление подсети называется **маской подсети**. На следующем рисунке приведен пример упомянутых адресов:

IP-адрес (192.168.1.53)	
Адрес сети	Адрес узла
192.168.1	53

Рис. 7.6. Подсеть с адресами сети и узла

Предположим, что в нашем адресе IPv4 (192.168.1.53) адрес сети — это 192.168.1, а адрес узла — 53. В этом случае маска подсети выглядит так:

192.168.1.0

Чтобы составить эту маску, мы отбросили наименьшую значащую группу полного адреса, которая представляла адрес узла (53), и заменили ее на 0. Этот 0 означает начальный адрес в подсети. Другими словами, в подсети допускаются узлы с любым адресом в диапазоне от 0 до 255. Например, 192.168.1.92 — это действительный (и распространенный) IP-адрес в сети 192.168.1.0.

Подсеть также можно описать с помощью нотации **CIDR**², в которой сначала указывается префикс сети, а за ним следует косая черта (/) и длина префикса

¹ Классовая адресация в интернете столкнулась с теми же ограничениями, что первоначальная четырехуровневая схема, и вышла из массового употребления в 1990-х годах, сменившись бесклассовой адресацией (см. RFC 4632). Однако рудименты прежней системы сохранились в некоторых аппаратных и программных решениях, в частности в стандартной конфигурации масок подсетей. — *Примеч. ред.*

² Сокр. Classless Inter-Domain Routing (бесклассовая междоменная маршрутизация). — *Примеч. пер.*

(в битах). В нашем случае нотация CIDR для подсети **192.168.1.0** выглядит так:

192.168.1.0/24

Первые три группы в сетевом адресе занимают $3 \times 8 = 24$ бита, отсюда и обозначение /24.

Когда проектируют подсети, обычно начинают с того, что задают адрес сети. Возвращаясь к нашему примеру, предположим, что мы хотим, чтобы адреса узлов в сети начинались со **100** и заканчивались **125**. Давайте посмотрим, как этого добиться.

Посмотрите на двоичное представление адреса **192.168.1.100**:

11000000.10101000.00000001.01100100

Последняя выделенная группа представляет здесь адрес узла (**100**). Чтобы адреса узлов начинались со **100**, нужно запретить адреса до **99** включительно. Ближайшим круглым двоичным числом будет $64 + 32 = 96$, а эквивалентное двоичное значение выглядит так:

01100000

Другими словами, три старших бита в адресе узла зарезервированы. В маске подсети им будут соответствовать единицы. Три этих бита будут добавлены к уже зарезервированным 24 битам сетевого адреса (**192.168.1**), поэтому в маске будет $24 + 3 = 27$ бит:

11111111.11111111.11111111.11100000

В десятичном виде итоговая сетевая маска выглядит так:

255.255.255.224

А вот нотация CIDR для соответствующей подсети:

192.168.1.96/27¹

В оставшихся пяти битах может уместиться $2^5 = 32$ возможных адреса узлов, начиная с **97**, так что максимальный адрес узла составит $96 + 32 - 1 = 127$ (мы вычли 1, потому что начальный адрес **97** входит в 32 доступных адреса).

В этом диапазоне из 32 адресов последний IP-адрес (**192.168.1.127**) резервируется в качестве **широковещательного**. Таким образом, если не учитывать

¹ Обратите внимание, что подсети **192.168.1.96/27** соответствует диапазон адресов узлов от **96** до **127**, а не от **100** до **125**, как было заявлено изначально. Для исходного диапазона адресов (от **100** до **125**) понадобится 5 записей в формате CIDR (а именно: **192.168.1.100/30**, **192.168.1.104/29**, **192.168.1.112/29**, **192.168.1.120/30**, **192.168.1.124/31**). — Примеч. ред.

широковещательный адрес, то в нашем примере максимальным адресом узла в подсети будет **192.168.1.126**.

Подробнее о подсетях вы можете узнать в *RFC 1918* (tools.ietf.org/html/rfc1918). Раз уж мы упомянули широковещательные адреса, давайте вкратце о них поговорим.

Широковещательные адреса

Широковещательный адрес — это зарезервированный IP-адрес в сети или подсети, с помощью которого можно передать массовое сообщение (то есть порцию данных) всем устройствам в сети. Широковещательным адресом становится последний IP-адрес в сети или подсети, когда это возможно.

Например, в сети **192.168.1.0/24** широковещательным адресом будет **192.168.1.255**. А в примере из предыдущего раздела фигурировала подсеть **192.168.1.96/27**, широковещательный адрес которой — **192.168.1.127**.

Адреса IPv6

Адрес IPv6 — это 128-битное (16-байтовое) число, которое обычно выражается в виде восьми групп из чисел по 2 байта (16 бит), разделенных двоеточием (:). В каждой группе может быть шестнадцатеричное значение от **0000** до **FFFF**. Вот пример адреса IPv6:

```
2001:0b8d:8a52:0000:0000:8b2d:0240:7235
```

Этот адрес можно представить в сокращенной форме:

```
2001:b8d:8a52::8b2d:240:7235
```

Во втором представлении ведущие нули в каждой группе опускаются, а две группы из нулей (**0000:0000**) сворачиваются в пустую группу (::).

Подсети IPv6 обозначаются по тому же принципу, что подсети IPv4. Например, **2001:b8d:8a52::/64** — это подсеть с 64-битным префиксом, в которую входит примерно 18×10^{18} адресов узлов. Для IPv6 длина префикса выражается в виде целого числа от 1 до 128.

Чтобы подробнее узнать об IPv6, можно обратиться к *RFC 8200* (datatracker.ietf.org/doc/html/rfc8200).

Теперь, когда вы знакомы с IP-адресами, самое время поговорить о некоторых связанных с ними сетевых конструкциях, которые служат программной реализацией IP-адресов, а именно о сокетах и портах.

Сокеты и порты

Сокет — это программная структура данных, которая представляет сетевой узел таким образом, чтобы к нему можно было обращаться для обмена данными.

Хотя это понятие скорее из области программирования, тем не менее в Linux **сетевой сокет** — это всего лишь файловый дескриптор, которым можно управлять через сетевой **API**. Прикладные процессы используют сокеты, чтобы передавать и принимать данные. Сокеты можно программно создавать и удалять. Сокет остается активным (то есть может отправлять или получать данные) только в пределах жизненного цикла процесса, который его создал.

Сетевые сокеты работают на *транспортном уровне* в модели OSI. В сокетном соединении есть две конечные точки — отправитель и получатель, и у каждого из них свой IP-адрес. Следовательно, в структуре данных сокета одним из ключевых полей будет *IP-адрес* конечной точки, которой принадлежит сокет.

Обе конечные точки создают свои сокеты и управляют ими через сетевые процессы, которые используют эти сокеты. Отправитель и получатель могут установить между собой несколько соединений, чтобы обмениваться данными, и некоторые из этих соединений могут даже работать параллельно. Как отличать эти соединения друг от друга? Одних IP-адресов тут недостаточно, и именно здесь в дело вступают **порты**.

Сетевой порт — это логическая конструкция, которая позволяет идентифицировать определенный процесс или сетевую службу, запущенную на узле. Порт — это целое число в диапазоне от 0 до 65535. Обычно порты в диапазоне от 0 до 1024 назначаются наиболее часто используемым службам в системе и называются **общезвестными портами**. Вот несколько примеров общезвестных портов и соответствующих сетевых служб:

21	FTP
22	SSH
25	SMTP
53	DNS
67	Сервер DHCP
68	Клиент DHCP
80	HTTP
443	HTTPS

Номера портов выше 1024 предназначены для общего использования и обычно служат **динамическими** портами.

Порт всегда связан с IP-адресом. В конечном счете сокет — это комбинация IP-адреса и порта (например, 192.168.1.53:443). Историческое определение сокета находится в *RFC 147* (<https://tools.ietf.org/html/rfc147>), а список общезвестных портов поддерживается по адресу iana.org/assignments/service-names-port-numbers.

Теперь давайте применим полученные знания на практике и научимся настраивать локальный сетевой стек в Linux.

Как настраивать сетевое окружение в Linux

В этом разделе описывается **настройка сети** TCP/IP для платформ Ubuntu и Fedora на примере их последних версий на момент написания этой книги. Все описанные принципы применимы и к большинству остальных дистрибутивов Linux, хотя некоторые утилиты и конфигурационные файлы могут отличаться.

Чтобы получить текущие IP-адреса системы, можно запустить утилиту командной строки `ip` с подкомандой `addr`:

```
packt@neptune:~$ ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group
default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
group default qlen 1000
    link/ether 52:54:00:d6:7d:69 brd ff:ff:ff:ff:ff:ff
    inet 192.168.122.117/24 metric 100 brd 192.168.122.255 scope global dynamic
    enp1s0
        valid_lft 3071sec preferred_lft 3071sec
    inet6 fe80::5054:ff:fed6:7d69/64 scope link
        valid_lft forever preferred_lft forever
```

Мы выделили здесь такие важные сведения, как идентификатор сетевого интерфейса (2: `enp1s0`) и подсеть IPv4 (192.168.122.117/24).

Далее мы рассмотрим сетевую конфигурацию Ubuntu. На момент написания книги самая свежая версия этой ОС — Ubuntu 22.04.2 LTS.

Как настраивать сеть в Ubuntu

В Ubuntu 22.04 сеть можно довольно просто настроить с помощью утилиты командной строки `netplan`, в которой привязки сетевых интерфейсов задаются с помощью конфигурационного файла в формате **YAML**. Конфигурация `netplan` находится в каталоге `/etc/netplan/`, содержимое которого можно просмотреть с помощью такой команды:

```
ls /etc/netplan/
```

В нашем случае конфигурационный файл называется `00-installer-config.yaml`, и его понадобится редактировать, чтобы изменить настройки сети. Перед тем как вносить изменения в файл, всегда имеет смысл создать резервную копию.

Одна из типичных задач настройки сети заключается в том, чтобы задать либо динамический, либо статический IP-адрес. В следующих разделах мы покажем, как сделать и то и другое, начиная с динамического адреса.

Как настроить динамический IP-адрес

Чтобы включить динамический IP-адрес, который выделяется с помощью DHCP, нужно отредактировать конфигурационный файл `netplan` и установить атрибут `dhcp4` в значение `true` для нужного сетевого интерфейса (в нашем случае `enp1s0`). Откройте файл с помощью текстового редактора (например, `nano`):

```
sudo nano /etc/netplan/00-installer-config.yaml
```

В следующем листинге выделена строка, в которую нужно внести изменения:

```
network:
  ethernets:
    enp1s0:
      dhcp4: true
version: 2
```

Сохранив конфигурационный файл, можно проверить наши изменения с помощью подкоманды `try`:

```
packt@neptune:~$ sudo netplan try
Do you want to keep these settings?
# Хотите ли вы сохранить эти настройки?
Press ENTER before the timeout to accept the new configuration
# Чтобы подтвердить новую конфигурацию,
# нажмите Enter до истечения времени ожидания
Changes will revert in 116 seconds
# Изменения будут отменены через 116 с
Configuration accepted.
# Конфигурация подтверждена
```

Подкоманда `try` проверяет новую конфигурацию и предлагает пользователю подтвердить изменения, а подкоманда `apply` применяет их к системе:

```
sudo netplan apply
```

Далее мы настроим статический IP-адрес с помощью `netplan`.

Как настроить статический IP-адрес

Чтобы задать статический IP-адрес сетевого интерфейса (например, `192.168.122.22` с маской подсети `/24`), отредактируйте конфигурационный файл `netplan` таким образом:

```
network:
  ethernets:
    enp1s0:
      dhcp4: false
      addresses:
        - 192.168.122.22/24
      gateway4: 192.168.122.1
```

```
nameservers:
  addresses: [8.8.8.8, 8.8.4.4]
version: 2
```

После того как вы сохраните конфигурацию, ее можно проверить и подтвердить, а затем применить изменения так же, как для динамического IP-адреса:

```
sudo netplan try
sudo netplan apply
```

Дополнительные сведения о `netplan` можно найти во встроенной справке (`netplan --help`) или на соответствующей справочной странице (`man netplan`).

Далее мы рассмотрим сетевую конфигурацию Fedora. На момент написания этой книги последняя версия этой ОС — Fedora 37.

Как настраивать сеть в Fedora/RHEL

Начиная с Fedora 33 и RHEL 9, конфигурационные файлы сети *больше не хранятся* в каталоге `/etc/sysconfig/network-scripts/`. Чтобы настраивать сеть в Fedora/RHEL, рекомендуется использовать утилиту `nmcli`, которая управляет демоном сетевого окружения `NetworkManager`. В новых версиях ОС его конфигурационные файлы хранятся в каталоге `/etc/NetworkManager/system-connections/`.

Давайте просмотрим сведения о сетевых соединениях с помощью базовых команд `nmcli`. Во-первых, запустим `nmcli connection`, чтобы получить информацию о нашем активном соединении:

```
[packt@saturn ~]$ nmcli connection
NAME                                UUID                                TYPE      DEVICE
Wired connection 1                  ce007bd0-cc92-3a30-a367-95ecbf728ab7 ethernet  enp1s0
```

Здесь отображаются основные сведения об имени соединения, его UUID, типе и используемом устройстве.

Как и в случае с Ubuntu, настраивать конфигурацию сети в Fedora/RHEL часто приходится для того, чтобы задать либо динамический, либо статический IP-адрес. В следующих разделах мы покажем, как сделать и то и другое.

Как настроить динамический IP-адрес

Чтобы настроить динамический IP-адрес с помощью `nmcli`, запустите такую команду:

```
sudo nmcli connection modify 'Wired connection 1' ipv4.method auto
```

Директива `ipv4.method auto` включает DHCP. Эта команда ничего не выводит; после того как она выполнится, вы просто вернетесь к приглашению командной строки. Чтобы проверить, сработала ли она, можно просмотреть каталог `/etc/NetworkManager/system-connections/`. В нашем случае там находится новый

конфигурационный файл, который называется так же, как наше соединение. Вот его фрагмент:

```
[packt@saturn ~] $ sudo cat /etc/NetworkManager/system-connections/Wired\
connecti on\ 1.nmconnection
[connection]
id=Wired connection 1
uuid=ce007bd0-cc92-3a30-a367-95ecbf728ab7
type=ethernet
autoconnect-priority=-999
interface-name=enp1s0
timestamp=1679137326

[ethernet]

[ipv4]
method=auto

[ipv6]
addr-gen-mode=default
method=auto
```

Теперь давайте настроим статический IP-адрес.

Как настроить статический IP-адрес

Чтобы настроить статический IP-адрес с помощью `nmcli`, нужно выполнить несколько команд. Прежде всего, надо задать сам адрес:

```
sudo nmcli connection modify 'Wired connection 1' ipv4.address 192.168.122.3/24
```

Если до этого статический IP-адрес не настраивался, рекомендуется сохранить изменения, прежде чем переходить к следующим шагам. Для этого перезапустите сетевой интерфейс `'Wired connection 1'`:

```
sudo nmcli connection down 'Wired connection 1'
sudo nmcli connection up 'Wired connection 1'
```

Далее нужно задать IP-адреса шлюза и DNS:

```
sudo nmcli connection modify 'Wired connection 1' ipv4.gateway 192.168.122.1
sudo nmcli connection modify 'Wired connection 1' ipv4.dns 8.8.8.8
```

Наконец, надо отключить DHCP с помощью такой команды:

```
sudo nmcli connection modify 'Wired connection 1' ipv4.method manual
```

После этих изменений снова перезапустите сетевой интерфейс с помощью тех же директив `up` и `down`.

Давайте посмотрим, к какому результату привели все эти команды. После того как соединение восстановится, можно проверить новый IP-адрес и конфигурационный файл. Наш новый адрес (`192.168.122.3`) отобразится в выводе `ip addr`:

```
2: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
group default qlen 1000
    link/ether 52:54:00:37:a2:73 brd ff:ff:ff:ff:ff:ff
    inet 192.168.122.3/24 brd 192.168.122.255 scope global noprefixroute enp1s0
        valid_lft forever preferred_lft forever
    inet6 fe80::d78a:3dd9:7af7:c8d2/64 scope link noprefixroute
        valid_lft forever preferred_lft forever
```

Теперь давайте посмотрим, что изменилось в конфигурационном файле после того, как мы настроили статический IP-адрес. Помните, что файл находится в каталоге `/etc/NetworkManager/system-connections/`:

```
[connection]
id=Wired connection 1
uuid=ce007bd0-cc92-3a30-a367-95ecbf728ab7
type=ethernet
autoconnect-priority=-999
interface-name=enp1s0
timestamp=1679147993
```

```
[ethernet]
```

```
[ipv4]
address1=192.168.122.3/24,192.168.122.1
dns=8.8.8.8;
method=manual
```

`nmcli` — мощная и полезная утилита. В конце этой главы мы приведем ссылку на издание, из которого можно узнать о ней подробнее. А сейчас посмотрим, как настраивать сетевые службы в openSUSE.

Как настраивать сеть в openSUSE

В openSUSE можно настраивать сеть с помощью служб **Wicked** или **NetworkManager**. Как правило, **NetworkManager** по умолчанию запускается на рабочих станциях, а **Wicked** — на серверах openSUSE.

Например, чтобы посмотреть, какая служба работает по умолчанию в openSUSE Leap, можно запустить такую команду:

```
sudo systemctl status network
```

В нашем случае (на ноутбуке) видно, что запущен **NetworkManager**:

```
alexandru@localhost:~> sudo systemctl status network
● NetworkManager.service - Network Manager
   Loaded: loaded (/usr/lib/systemd/system/NetworkManager.service; enabled; v>
   Drop-In: /usr/lib/systemd/system/NetworkManager.service.d
            └─NetworkManager-ovs.conf
   Active: active (running) since Sun 2023-03-19 12:06:46 EET; 33min ago
     Docs: man: NetworkManager (8)
   Main PID: 1387 (NetworkManager)
```

Если запустить ту же команду на сервере openSUSE Leap под управлением виртуальной машины, результат отличается. Как видите, по умолчанию запущен Wicked:

```
packt@localhost:~> sudo systemctl status network
● wicked.service - wicked managed network interfaces
   Loaded: loaded (/usr/lib/systemd/system/wicked.service; enabled; vendor pr>
   Active: active (exited) since Sun 2023-03-19 13:42:07 EET; 2min 18s ago
   Main PID: 769 (code=exited, status=0/SUCCESS)
```

Мы будем выполнять все примеры этого раздела в виртуальной машине с серверной версией openSUSE Leap и настраивать сеть с помощью Wicked. Давайте настроим динамический IP-адрес.

Как настроить динамический IP-адрес

Прежде чем что-то настраивать, давайте посмотрим на активные подключения и устройства. Это можно сделать с помощью такой команды:

```
sudo wicked show all
```

Вы увидите все активные устройства. Вот фрагмент вывода на нашем компьютере:

```
eth0 up
    link:      #2, state up, mtu 1500
    type:      ethernet, hwaddr 52:54:00:dc:4e:e0
    config:    compat:suse:/etc/sysconfig/network/ifcfg-eth0
    leases:    ipv4 dhcp granted
    leases:    ipv6 dhcp requesting
    addr:      ipv4 192.168.122.146/24 [dhcp]
    route:     ipv4 default via 192.168.122.1 proto dhcp
```

Интерфейс `eth0` — это порт Ethernet. Команда также выводит информацию о локальной петле (`lo`), которую мы здесь опустили.

Wicked хранит конфигурационные файлы в каталоге `/etc/sysconfig/network`. Если его просмотреть, можно увидеть много конфигурационных файлов для существующих соединений:

```
packt@localhost:~> ls -la /etc/sysconfig/network/
total 64
drwxr-xr-x 1 root root 176 Mar 3 21:00 .
drwxr-xr-x 1 root root 584 Mar 11 14:18 ..
-rw-r--r-- 1 root root 9691 Mar 3 21:00 config
-rw-r--r-- 1 root root 15678 Mar 3 21:00 dhcp
-rw----- 1 root root 46 Mar 3 21:00 ifcfg-eth0
-rw-r--r-- 1 root root 34 Mar 3 21:00 ifcfg-eth0.bak
-rw----- 1 root root 147 Mar 3 21:00 ifcfg-lo
-rw-r--r-- 1 root root 21738 Aug 1 2022 ifcfg.template
drwxr-xr-x 1 root root 0 Mar 15 2022 if-down.d
drwxr-xr-x 1 root root 0 Mar 15 2022 if-up.d
drwx----- 1 root root 0 Mar 15 2022 providers
drwxr-xr-x 1 root root 38 Mar 3 20:59 scripts
```

В нашем случае (а может быть, и в вашем) нужный файл называется `ifcfg-eth0`. Давайте посмотрим на его содержимое:

```
BOOTPROTO='dhcp'
STARTMODE='auto'
ZONE=public
```

Это верная, но довольно скудная информация. Чтобы просмотреть свойства соединений подробнее, можно запустить такую команду:

```
sudo wicked show-config
```

Вот фрагмент вывода этой команды с подробными сведениями об IPv4:

```
<ipv4>
  <enabled>true</enabled>
  <arp-verify>true</arp-verify>
</ipv4>
<ipv4:dhcp>
  <enabled>true</enabled>
  <flags>group</flags>
  <update>default-route,hostname,dns,nis,ntp,nds,mtu,tz,boot</update>
  <defer-timeout>15</defer-timeout>
  <recover-lease>true</recover-lease>
  <release-lease>false</release-lease>
</ipv4:dhcp>
```

Чтобы лучше разобраться в этих сведениях, просмотрите файлы `config` и (если есть) `dhcp` в каталоге `/etc/sysconfig/network`. В них задаются специфические переменные и параметры по умолчанию, которые необходимы, чтобы настраивать сетевые устройства.

Динамические IP-адреса обычно настраиваются по умолчанию при установке операционной системы. Если у вас такой адрес не настроен, то все, что вам нужно сделать, — это создать конфигурационный файл в каталоге `/etc/sysconfig/network` и назвать его в соответствии с именем устройства, через которое вы подключаетесь к сети, например `ifcfg-eth0`. В этот файл нужно вставить всего три строки, которые показаны на с. 257. Вот какие команды помогут выполнить эту задачу:

- Проверьте имя устройства с помощью команды `ip addr`:
`ip addr show`
- Создайте конфигурационный файл в каталоге `/etc/sysconfig/network`:
`sudo nano /etc/sysconfig/network/ifcfg-eth0`
- Наберите нужный код для настройки DHCP:

```
BOOTPROTO='dhcp'
STARTMODE='auto'
ZONE=public
```

- Перезапустите службу Wicked:
`sudo systemctl restart wicked`
- Проверьте соединение с помощью браузера или команды `ping`, например:
`ping google.com`

В следующем разделе мы покажем, как настроить статический IP-адрес.

Как настроить статический IP-адрес

Чтобы настроить статическую конфигурацию IP, нужно вручную задать переменные в тех же конфигурационных файлах, которые рассматривались в предыдущем разделе и размещаются в каталоге `/etc/sysconfig/network`. Например, можно создать новый файл для устройства `eth0` и настроить в нем свойства соединения. Мы рассмотрим эту задачу на примере серверной версии openSUSE Leap, которая работает в виртуальной машине. Однако сначала мы советуем вам просмотреть справочную страницу утилиты `ifcfg`, потому что там вы найдете немало ценной информации о переменных, которые мы будем использовать.

Итак, вот что можно сделать, чтобы настроить нужную конфигурацию:

- Сначала проверьте IP-адрес и имя сетевого устройства. В нашем случае динамически выделяемый IP-адрес — `192.168.122.146`, а имя устройства — `eth0`:

```
2: eth0: <BROADCAST, MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state
UP group default qlen 1000
    link/ether 52:54:00:dc:4e:e0 brd ff:ff:ff:ff:ff:ff
    altname enp1s0
    inet 192.168.122.146/24 brd 192.168.122.255 scope global eth0
        valid_lft forever preferred_lft forever
    inet6 fe80::5054:ff:fedc:4ee0/64 scope link
        valid_lft forever preferred_lft forever
```

- Перейдите в каталог `/etc/sysconfig/network` и отредактируйте файл `ifcfg-eth0`. Чтобы настроить статический IP-адрес, введите такие пять строк:

<code>BOOTPROTO='static'</code>	Переключиться в режим фиксированного IP-адреса
<code>STARTMODE='auto'</code>	Автоматически включать сетевой интерфейс при загрузке
<code>IPADDR='192.168.122.144'</code>	IP-адрес, который мы выбираем для машины
<code>ZONE='public'</code>	Зона для утилиты <code>firewalld</code>
<code>PREFIXLEN='24'</code>	Длина префикса подсети

- Сохраните изменения в файле и перезапустите демон Wicked:

```
sudo systemctl restart wickedd.service
```


- Включите интерфейс, чтобы применить изменения:

```
sudo wicked ifup eth0
```

- Снова запустите команду `ip addr`, чтобы убедиться, что отображается новый IP-адрес:

```
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
group default ql en 1000
    link/ether 52:54:00:dc:4e:e0 brd ff:ff:ff:ff:ff:ff
    altname enp1s0
    inet 192.168.122.146/24 brd 192.168.122.255 scope global eth0
        valid_lft forever preferred_lft forever
    inet 192.168.122.144/24 brd 192.168.122.255 scope global secondary eth0
        valid_lft forever preferred_lft forever
    inet6 fe80::5054:ff:fedc:4ee0/64 scope link
        valid_lft forever preferred_lft forever
```

Теперь вы знаете, как настраивать сетевые устройства во всех основных дистрибутивах Linux с помощью предпочтительных утилит. Мы рассмотрели этот вопрос лишь поверхностно, но постарались обеспечить достаточно информации для того, чтобы вы могли приступить к работе с сетевыми интерфейсами в Linux. Чтобы изучить эту тему подробнее, почаще читайте справочные страницы, которые поставляются вместе с вашей операционной системой. В следующем разделе мы рассмотрим, как настраивать имена узлов.

Как настраивать имена узлов

Чтобы получить текущее имя узла на машине Linux, можно запустить команду `hostname` или `hostnamectl` без аргументов, например:

```
hostname
```

Самый удобный способ изменить имя узла — это команда `hostnamectl` с подкомандой `set-hostname`. Давайте с ее помощью изменим имя узла на `earth`:

```
sudo hostnamectl set-hostname earth
```

После этого можно запустить `hostnamectl` без аргументов, чтобы проверить новое имя. Эта команда выводит более подробные сведения, чем `hostname`:

```
packt@neptune:~$ hostname
neptune
```

```
packt@neptune:~$ sudo hostnamectl set-hostname earth
```

```
packt@neptune:~$ hostname
earth
```

```
packt@neptune:~$ hostnamectl
Static hostname: earth
Iconname: computer-vm
Chassis: vm
```

```
Machine ID: 7c7f082fb11d431a9823a11ba9e87ad0
Boot ID: f7eb64629ddf4424836760dfab2e9146
Virtualization: kvm
Operating System: Ubuntu 22.04.2 LTS
Kernel: Linux 5.15.0-67-generic
Architecture: x86-64
Hardware Vendor: QEMU
Hardware Model: Standard PC _Q35 + ICH9, 2009_
```

С помощью `hostname` также можно *временно* изменить имя узла, например:

```
sudo hostname jupiter
```

Однако это изменение не сохранится после перезагрузки, если аналогично не отредактировать имя узла в файлах `/etc/hostname` и `/etc/hosts`. Вот как можно изменить имя с `earth` обратно на `neptune`:

```
packt@neptune:~$ hostname
earth
```

```
packt@neptune:~$ sudo hostname neptune
```

```
packt@neptune:~$ cat /etc/hostname
earth
```

```
packt@neptune:~$ sudo nano /etc/hostname
# Открываем файл /etc/hostname в текстовом редакторе nano
# и меняем в нем earth на neptune
```

```
packt@neptune:~$ sudo nano /etc/hosts
# Аналогично редактируем файл /etc/hosts
```

```
packt@neptune:~$ hostname
neptune
```

После того как вы настроили имя узла, может понадобиться выйти из системы и снова войти в нее, чтобы изменения отразились в командной строке. Продуманные имена узлов помогают эффективно управлять сетевым окружением. В следующем разделе вы узнаете о сетевых службах в Linux.

Как работать с сетевыми службами в Linux

В этом разделе мы перечислим несколько наиболее распространенных сетевых служб, которые бывают в Linux. Некоторые из них могут быть не включены или даже не установлены по умолчанию в вашем дистрибутиве. В главе 9 «Как обеспечивать безопасность в Linux» и в главе 10 «Аварийное восстановление, диагностика и устранение неполадок» мы подробнее поговорим о том, как установить и настроить некоторые из этих служб. А в этом разделе мы сосредоточимся на том, что они собой представляют, как они работают и какие сетевые протоколы используют для обмена данными.

Сетевая служба — это, как правило, системный процесс, который реализует функциональность прикладного уровня (уровень 7 в модели OSI), чтобы передавать и/или принимать данные. Для сетевых служб обычно характерна одноранговая или клиент-серверная архитектура.

В **одноранговой сети** каждый узел запускает свой собственный экземпляр сетевой службы с такими же привилегиями, как у других узлов. Узлы совместно обращаются к общему набору данных и обмениваются этими данными: например, в сети серверов DNS каждый сервер наравне с другими использует и обновляет записи доменных имен.

В сеть типа **«клиент — сервер»** обычно входит один или несколько серверных узлов и несколько клиентов, каждый из которых обращается к одному или нескольким серверам. Пример такой сетевой службы — SSH: клиент SSH подключается к удаленному серверу SSH через защищенный сеанс терминала — возможно, для того, чтобы дистанционно администрировать систему.

В каждом из следующих подразделов кратко описана та или иная сетевая служба. Чтобы подробнее познакомиться с этими службами, мы рекомендуем вам уделить внимание главе 13 или изданиям, которые рекомендуются в конце текущей главы. Начнем с серверов DHCP.

Серверы DHCP

Сервер DHCP использует протокол DHCP для того, чтобы устройства в сети могли запрашивать IP-адреса, которые назначены динамически. Мы кратко упоминали про DHCP в разделе «Протоколы TCP/IP» ранее в этой главе.

Компьютер или устройство, которое запрашивает услугу DHCP, передает широковещательное сообщение (запрос) в сеть, чтобы найти сервер DHCP, который, в свою очередь, предоставляет нужный IP-адрес и другую информацию.

В протоколе DHCP начальная процедура *обнаружения* между клиентом и сервером происходит на канальном уровне (уровень 2) в модели OSI. Поскольку на этом уровне в качестве единиц данных используются сетевые кадры, пакеты обнаружения DHCP не могут выйти за пределы локальной сети. Другими словами, клиент DHCP может инициировать связь только с *локальным* сервером DHCP.

После начального *квитирования* (на уровне 2) DHCP переходит на транспортный протокол UDP, передавая дейтаграммы через сокеты (уровень 4). Поскольку UDP — это протокол без установки соединения, клиент и сервер DHCP передают друг другу сообщения, предварительно не налаживая сеанс связи. Значит, и клиенту, и серверу нужны общеизвестные порты DHCP, через которые они могли бы обмениваться данными. Эти порты — 68 (для сервера DHCP) и 67 (для клиента DHCP).

Сервер DHCP хранит совокупность IP-адресов и другие данные о конфигурации клиента (например, MAC-адреса и адреса доменных серверов) для каждого устройства в сети, которое запрашивает услугу DHCP.

Чтобы динамически назначать IP-адреса, серверы DHCP используют механизм **аренды**. Каждый адрес выделяется на определенный **срок аренды**, который может быть конечным или бесконечным. Когда этот срок истекает, сервер DHCP может переназначить адрес другому клиенту по его запросу. Устройство может удерживать свой динамический IP-адрес, регулярно запрашивая у сервера **продление аренды**. Если этого не делать, то устройство может лишиться динамического IP-адреса. Тогда в результате нового запроса DHCP оно может получить новый IP-адрес, если прежний адрес сервер уже выделил кому-то еще.

Простой способ запросить данные о сервере DHCP — команда `ip route`:

```
packt@neptune:~$ ip route
default via 192.168.122.1 dev enp1s0 proto dhcp src 192.168.122.117 metric 100
192.168.122.0/24 dev enp1s0 proto kernel scope link src 192.168.122.117 metric 100
192.168.122.1 dev enp1s0 proto dhcp scope link src 192.168.122.117 metric 100
```

Адрес сервера DHCP (в данном случае 192.168.122.1) указан в первой строке.

В главе 13 «Как настраивать серверы Linux» мы рассмотрим практические подробности того, как устанавливать и настраивать сервер DHCP. Подробнее о DHCP можно узнать в *RFC 2131* (<https://tools.ietf.org/html/rfc2131>).

Серверы DNS

Сервер DNS осуществляет разрешение имен, то есть преобразует доменное имя узла (например, `wikipedia.org`) в IP-адрес (например, `208.80.154.224`). Эта процедура происходит по протоколу DNS, который кратко описывался в разделе «Протоколы TCP/IP» ранее в этой главе. Если в сети TCP/IP функционирует служба DNS, то компьютеры и другие устройства могут идентифицировать друг друга и обращаться друг к другу по именам узлов, а не только по IP-адресам.

Система DNS во многом напоминает адресную книгу. Имена узлов относительно легче запомнить, чем IP-адреса. Даже в локальной сети, где подключено всего несколько компьютеров и устройств, было бы неудобно каждый раз обращаться к тому или иному узлу по IP-адресу (равно как и запоминать их). Интернет опирается на глобально распределенную сеть серверов DNS.

Серверы DNS выступают в четырех качествах: **рекурсивные**, **корневые**, **серверы доменов верхнего уровня** и **авторитетные**. Все эти типы серверов работают вместе, чтобы предоставить вам интернет в той форме, в которой вы видите его в своем браузере.

Рекурсивный сервер DNS — это «разрешатель», который помогает найти IP-адрес нужного сайта. Когда вы инициируете поиск, рекурсивный сервер подключается к другим серверам DNS, чтобы найти нужный адрес и вернуть его

вам в виде сайта. Рекурсивный поиск DNS выполняется быстрее итеративного, потому что кэширует каждый запрос. При рекурсивном типе запроса сервер DNS обращается прежде всего к самому себе, но при этом также отправляет запрос на другой сервер DNS.

В отличие от этого, **итеративный поиск** DNS выполняется каждым сервером DNS напрямую, без кэширования. Например, при итеративном запросе каждый сервер DNS возвращает адрес другого сервера DNS до тех пор, пока один из них не найдет подходящий IP-адрес для заданного имени хоста и не ответит клиенту. Чтобы подробнее узнать о типах серверов DNS, ознакомьтесь с учебным пособием Cloudflare: cloudflare.com/learning/dns/what-is-dns.

Серверы DNS поддерживают (а также могут предоставлять для совместного использования) набор файлов баз данных, которые также известны как **файлы зон** и обычно представляют собой простые текстовые файлы в кодировке ASCII, в которых хранится сопоставление имен и IP-адресов. В Linux одним из таких файлов является `/etc/resolv.conf`.

Чтобы узнать, какой сервер DNS обслуживает ваш локальный компьютер, можно обратиться к файлу `/etc/resolv.conf` с помощью конвейера `cat | grep`:

```
packt@neptune:~$ cat /etc/resolv.conf | grep nameserver
nameserver 127.0.0.53
```

Простой способ запросить данные о сервере имен для произвольного узла в сети — использовать утилиту `nslookup`. Чтобы установить ее в Ubuntu/Debian, запустите такую команду:

```
sudo apt install bind9-dnsutils
```

В Fedora `nslookup` устанавливается так:

```
sudo dnf install bind9-next-utils
```

Теперь, чтобы узнать, какой сервер имен обслуживает компьютер `neptune.local` в нашей локальной сети, можно запустить такую команду:

```
packt@neptune:~$ nslookup neptune.local
Server:      127.0.0.53
Address:     127.0.0.53#53
Name:   neptune.local
Address: 192.168.122.117
```

Утилиту `nslookup` можно также использовать в интерактивном режиме. Например, чтобы запросить информацию о сервере имен для `wikipedia.org`, просто запустите `nslookup`, а затем в приглашении командной строки введите `wikipedia.org`:

```
packt@neptune:~$ nslookup
> wikipedia.org
Server: 127.0.0.53
Address: 127.0.0.53#53
```

```
Non-authoritative answer:
Name:   wikipedia.org
Address: 91.198.174.192
Name:   wikipedia.org
Address: 2620:0:862:ed1a::1
```

Чтобы выйти из интерактивного режима оболочки, введите команду `exit`. Вот краткая справка о том, что вывела команда `nslookup`:

Server / Address	Адрес петли обратной связи (127.0.0.53) и порт (53) локального сервера DNS
Name	Доменное имя, адрес которого мы ищем (wikipedia.org)
Address	Адреса IPv4 (91.198.174.192) и IPv6 (2620:0:862:ed1a::1), соответствующие доменному имени (wikipedia.org)

`nslookup` также может выполнять обратный поиск DNS, возвращая доменное имя по заданному IP-адресу. Следующая команда находит сервер имен (`dns.google`), который соответствует IP-адресу `8.8.8.8`:

```
packt@neptune:~$ nslookup 8.8.8.8
8.8.8.8.in-addr.arpa name=dns.google.
```

Authoritative answers can be found from:

Чтобы подробнее узнать об инструменте `nslookup`, обратитесь к соответствующей справочной странице (`man nslookup`).

Изучать информацию DNS также позволяет утилита `dig`. Как и `nslookup`, ее можно установить в составе пакета `bind9-dnsutils` на платформах Ubuntu/Debian или `bind9-next-utils` на платформе Fedora.

Например, следующая команда предоставляет информацию из доменной зоны для `google.com`:

```
dig google.com
```

Вывод может содержать несколько строк, самые важные из которых таковы:

```
;; ANSWER SECTION:
google.com.          39      IN      A       172.217.169.206
```

Чтобы выполнить обратный поиск по DNS с помощью `dig`, укажите ключ `-x`, а затем IP-адрес (например, `8.8.4.4`):

```
dig -x 8.8.4.4
```

Вот самое важное из того, что выводит эта команда:

```
;; ANSWER SECTION:
4.4.8.8.in-addr.arpa. 25900   IN      PTR     dns.google.
```

Чтобы получить дополнительную информацию об утилите **dig**, обратитесь к соответствующей справочной странице (**man dig**).

Протокол DNS работает на прикладном уровне (уровень 7) в модели OSI, а стандартный общеизвестный порт службы DNS — 53.

В главе 8 «Сценарии командной оболочки в Linux» мы подробнее поговорим о том, как на практике установить и настроить сервер DNS. Чтобы подробнее узнать о DNS, можно обратиться к *RFC 1035* (datatracker.ietf.org/doc/html/rfc1035).

Сетевые службы DHCP и DNS, пожалуй, теснее всего связаны со стеком TCP/IP и играют важнейшую роль при подключении компьютеров или устройств к сети. В конце концов, в сети нельзя обмениваться данными без правильной адресации и разрешения имен.

Однако распределенные сети и связанные с ними серверы приложений не ограничиваются чисто сетевым стеком под управлением серверов DNS и DHCP. В следующих разделах мы проведем краткий экскурс по некоторым наиболее важным серверам приложений, которые эксплуатируются в распределенных системах Linux.

Серверы аутентификации

В автономных системах Linux обычно применяется механизм аутентификации по умолчанию, когда учетные данные пользователей хранятся в локальной файловой системе (например, в файлах `/etc/passwd` и `/etc/shadow`). Мы рассмотрели соответствующие вопросы аутентификации в главе 4 «Как управлять пользователями и группами». Однако если нужно аутентифицировать пользователей за пределами локального компьютера — например, предоставлять им доступ к файловому или почтовому серверу, — то использовать одни и те же учетные данные на локальном и удаленном узлах было бы крайне нежелательно по соображениям безопасности.

В идеале нужно, чтобы в сети была централизованная оконечная точка аутентификации, которую обслуживает защищенный сервер аутентификации. Перед тем как пользователь получит доступ к ресурсам удаленной системы, учетные данные должны проверяться с помощью надежных механизмов шифрования.

Посмотрим, как устроен защищенный доступ к сетевому ресурсу на произвольном файловом сервере. Предположим, что пользователи аутентифицируются через службу **Active Directory**. Чтобы смонтировать запрошенный ресурс на локальном компьютере пользователя, понадобится ввести его учетные данные. Файловый сервер направляет запрос на аутентификацию от имени клиента на сервер аутентификации. Если аутентификация проходит успешно, клиент получает доступ к общему ресурсу сервера. На следующей диаграмме представлена простая последовательность операций удаленной аутентификации, в которой оконечной точкой аутентификации служит сервер **LDAP**:



Рис. 7.7. Процедура аутентификации с помощью LDAP

Вот некоторые стандартные механизмы защищенной аутентификации, доступные в Linux:

Kerberos	web.mit.edu/kerberos/
LDAP	redhat.com/en/topics/security/what-is-ldap-authentication
RADIUS	freeradius.org/documentation/
Диаметр	f5.com/glossary/diameter-protocol
TACACS+	https://datatracker.ietf.org/doc/rfc8907/

Сервер аутентификации LDAP в Linux можно настроить с помощью ПО OpenLDAP, которое рассматривалось в первом издании этой книги.

В этом разделе мы проиллюстрировали процедуру аутентификации на примере файлового сервера. Давайте продолжим эту тему и рассмотрим сетевые службы совместного доступа к файлам.

Совместный доступ к файлам

В общих сетевых терминах **совместный доступ к файлам** заключается в том, что клиентская машина может монтировать файловую систему, которая размещена на удаленном сервере, и обращаться к ней так, как если бы она была локальной. Приложения, которые работают на стороне клиента, получают доступ к общим файлам непосредственно на сервере. Например, файл с сервера можно открыть в текстовом редакторе, изменить, а затем сохранить обратно на сервере, причем все это будет происходить плавно и прозрачно. Технически реализовать такой удаленный доступ стало возможно благодаря службам и протоколам совместного доступа к файлам.

Для каждого сетевого протокола совместного доступа к файлам существует соответствующая клиент-серверная платформа. Хотя большинство сетевых файловых серверов (как и клиентов) реализованы для разных платформ, в следующих подразделах мы убедимся, что для определенных протоколов одни платформы подходят лучше, чем другие. Выбор между различными реализациями и протоколами файловых серверов в конечном итоге определяется требованиями совместимости, безопасности и производительности.

Вот несколько наиболее распространенных протоколов совместного доступа к файлам:

- **SMB** обеспечивает сетевое обнаружение и совместный доступ к файлам и принтерам, а также поддерживает межпроцессное взаимодействие по сети. SMB — относительно старый протокол, разработанный **корпорацией IBM** в 1980-х годах. В конце концов этим протоколом занялась компания Microsoft, которая внесла в него значительные изменения и выпустила в нескольких редакциях (вплоть до SMB 3.1.1).
- **CIFS** — одна из реализаций протокола SMB. Поскольку сами протоколы совпадают, клиенты SMB могут связываться с серверами CIFS, и наоборот. Хотя идиоматически SMB и CIFS одинаковы, у них существенно различается внутренняя реализация блокировки файлов и механизм пакетной обработки, а это в конечном итоге сказывается на производительности. В наше время CIFS используется редко, за исключением устаревших систем. SMB всегда предпочтительнее, особенно в свежих версиях SMB 2 или SMB 3.
- **Samba**, как и CIFS, — еще одна реализация SMB, которая предоставляет совместный доступ к файлам и принтерам для клиентов Windows на различных серверных платформах. Другими словами, клиенты Windows могут так же легко обращаться к файлам, каталогам и принтерам на сервере Samba, который установлен на Linux, как если бы они обращались к серверу Windows. Начиная с версии 4, Samba органически поддерживает домены Microsoft AD и Windows NT. По сути, сервер Samba под Linux может служить контроллером домена в сети AD под Windows. Следовательно, на сервере Linux можно не создавать учетные данные пользователей заново, а прозрачно использовать данные из домена Windows, а затем вручную синхронизировать их с учетными записями AD.
- **NFS** — этот протокол был разработан компанией Sun Microsystems и, по сути, действует по тому же принципу, что и SMB: предоставляет доступ к удаленным файлам так, как если бы они были локальными. NFS не совместим ни с CIFS, ни с SMB, а значит, клиенты NFS не могут напрямую обращаться к серверам SMB и наоборот.
- **AFP** — это проприетарный протокол совместного доступа к файлам, который создан компанией Apple и работает исключительно в сетевых средах macOS. Впрочем, помимо AFP, системы macOS поддерживают и стандартные протоколы совместного доступа к файлам, в том числе SMB и NFS.

В большинстве случаев в сетевом окружении Linux для совместного доступа к файлам используется протокол NFS, а для смешанных сетевых сред — например, таких, где разные узлы работают под управлением Windows, Linux и macOS, — лучше подходят Samba и SMB.

Некоторые протоколы совместного доступа к файлам (например, SMB) также поддерживают совместный доступ к принтерам и используются серверами печати. Далее мы подробнее рассмотрим совместный доступ к печати.

Серверы печати

Принт-сервер (или **сервер печати**) подключает принтер к клиентским узлам (компьютерам или мобильным устройствам) в сети с помощью протокола печати. Такие протоколы отвечают за задачи удаленной печати, например:

- обнаруживают принтеры или серверы печати;
- опрашивают состояние принтеров;
- передают, принимают, ставят в очередь или отменяют задания печати;
- запрашивают состояние заданий печати.

К распространенным протоколам печати относятся следующие:

- **LPD**;
- **протоколы общего назначения**, такие как SMB и TELNET;
- **технологии беспроводной печати**, например AirPrint от Apple;
- **протоколы онлайн-печати**, например Google Cloud Print.

Про упомянутый здесь протокол SMB (который также служит протоколом совместного доступа к файлам) мы уже говорили в разделе «Совместный доступ к файлам», а протокол связи TELNET рассматривается далее в разделе «Удаленный доступ».

Службы совместного доступа к файлам и принтерам предназначены в основном для того, чтобы разные компьютеры в сети могли совместно использовать или распечатывать документы. А когда нужно *передавать* документы с одного компьютера на другой, в дело вступают другие сетевые службы, которые реализуют передачу *файлов* и *электронной почты*. Далее мы рассмотрим передачу файлов.

Передача файлов

FTP — это стандартный сетевой протокол для передачи файлов между компьютерами в сети. Он работает в клиент-серверной среде, где клиент FTP инициирует удаленное соединение с сервером FTP, и файлы передаются в обоих направлениях. Протокол поддерживает одно **управляющее соединение** и одно

или несколько **соединений для передачи данных** между клиентом и сервером. Как правило, управляющее соединение устанавливается с портом 21 сервера FTP, и по нему передаются команды от клиента к серверу и обратно. Остальные соединения используются только для того, чтобы передавать данные, и согласовываются между клиентом и сервером через управляющее соединение. В этих соединениях обычно фигурируют эфемерные порты для входящего трафика, которые открыты только во время передачи данных и закрываются сразу после того, как она завершилась.

FTP согласовывает соединения для передачи данных в одном из двух режимов:

- **Активный режим.** Клиент FTP передает серверу команду **PORT**, сигнализируя о том, что клиент активно предоставляет номер входящего порта, через который будут передаваться данные.
- **Пассивный режим.** Клиент FTP передает серверу команду **PASV**, сигнализируя о том, что клиент пассивно ожидает от сервера номер порта для входящих соединений с данными.

В том, что касается конфигурации межсетевого экрана, FTP считается довольно запутанным протоколом из-за динамического характера соединений. Порт управляющего соединения обычно общеизвестен (например, 21 для незащищенного FTP), но соединения для передачи данных на передающем конце открываются на другом порте (обычно 20) с обеих сторон, а на принимающем конце входящие сокеты открываются в заранее настроенном диапазоне эфемерных портов (от 1024 до 65535).

Чаще всего FTP реализуется с дополнительным уровнем защиты одним из двух способов:

- **FTPS** (FTP через SSL). Соединение по FTP с шифрованием SSL/TLS. Управляющее соединение по умолчанию открывается на порте 990.
- **SFTP** (FTP через SSH). Управляющее соединение по умолчанию открывается на порте 22. Дополнительные сведения о протоколе SSH и подключении к серверу см. в разделе «Удаленный доступ» далее в этой главе.

Далее мы рассмотрим почтовые серверы и соответствующие протоколы обмена электронной почтой.

Почтовые серверы

Почтовый сервер (или **сервер электронной почты**) отвечает за доставку электронной почты по сети. Почтовый сервер может позволять пользователям обмениваться электронными письмами в пределах одной сети (например, компании или другой организации), а может передавать электронные письма на другие почтовые серверы за пределами локальной сети — например, через интернет.

Обмен электронными сообщениями обычно включает в себя несколько составляющих:

- **клиентское приложение** для работы с электронной почтой (например, Outlook или Gmail);
- один или несколько **почтовых серверов** (например, Exchange или Gmail);
- **корреспонденты**, которые обмениваются электронными сообщениями, — один отправитель и один или несколько получателей;
- **протокол электронной почты**, который управляет обменом данными между клиентским приложением и почтовыми серверами.

Самые популярные протоколы электронной почты — **POP3**, **IMAP** и **SMTP**. Давайте рассмотрим каждый из них подробнее.

POP3

POP3 — это стандартный протокол, который позволяет получать электронную почту и загружать ее с удаленного почтового сервера на локальный почтовый клиент. С помощью POP3 электронные письма можно читать в автономном режиме. После загрузки почта обычно удаляется с сервера POP3, что позволяет сэкономить место. Современные почтовые программы (Gmail, Outlook и др.) умеют сохранять копии писем на сервере: это важно для пользователей, которые обращаются к своей электронной почте из разных приложений на разных устройствах.

Порты POP3 по умолчанию:

- **110** — для незашифрованного соединения;
- **995** — для защищенного соединения с шифрованием SSL/TLS.

POP3 — относительно старый протокол электронной почты, который не всегда подходит для современных почтовых коммуникаций. Если пользователям нужно работать со своей почтой на нескольких устройствах, лучше выбрать IMAP, о котором мы сейчас поговорим.

IMAP

IMAP — это стандартный протокол для доступа к электронной почте на удаленном почтовом сервере. Этот протокол хранит письма на сервере, а их копия доступна для клиентов. Пользователь может обращаться к своей почте на нескольких устройствах, на каждом из которых установлено клиентское приложение IMAP.

Порты IMAP по умолчанию:

- **143** — для незашифрованного соединения;
- **993** — для защищенного соединения с шифрованием SSL/TLS.

POP3 и IMAP — стандартные протоколы для получения электронной почты, а для отправки используется SMTP.

SMTP

SMTP — это стандартный почтовый протокол для отправки электронных писем по внутренней сети или через интернет.

Порты SMTP по умолчанию:

- 25 — для незашифрованного соединения;
- 465 или 587 — для защищенного соединения с шифрованием SSL/TLS.

Если вы применяете или сами реализуете любой из стандартных почтовых протоколов, всегда лучше использовать соответствующую защищенную реализацию с самым современным шифрованием TLS, если это возможно. POP3, IMAP и SMTP также поддерживают аутентификацию пользователей, что создает дополнительный уровень безопасности — особенно если речь идет о коммерческих предприятиях или корпоративных средах.

Чтобы получить представление о том, как работает SMTP, давайте рассмотрим, как инициируется квити́рование с сервером SMTP службы Gmail от Google.

Прежде всего подключимся к этому серверу, используя защищенное (TLS) соединение, с помощью команды `openssl`:

```
openssl s_client -starttls smtp -connect smtp.gmail.com:587
```

Здесь мы вызвали команду `openssl`, использовали симуляцию клиента (`s_client`), запустили соединение SMTP с шифрованием TLS (`-starttls smtp`) и подключились к удаленному почтовому серверу Gmail на порте 587 (`-connect smtp.gmail.com:587`).

В данном случае сервер SMTP отвечает довольно длинным блоком квити́рования TLS, который заканчивается таким кодом:

```
---
No client certificate CA names sent
Peer signing digest: SHA256
Peer signature type: ECDSA
Server Temp Key: X25519, 253 bits
---
SSL handshake has read 4578 bytes and written 429 bytes
Verification: OK
---
New, TLSv1.3, Cipher is TLS_AES_256_GCM_SHA384
Server public key is 256 bit
Secure Renegotiation IS NOT supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
```

```
Early data was not sent
Verify return code: 0 (ok)
--
250 SMTPUTF8
```

Находясь в интерактивной подсказке команды `openssl`, мы инициируем соединение SMTP командой `HELO` (пишется именно так), которая *приветствует* сервер. Другой вариант этой команды — `EHLO`, и он используется для расширений службы ESMTP. Google ожидает следующее приветствие `HELO`:

```
HELO hellogoogle
```

Затем следует еще одно квитиование, которое заканчивается строкой `250 smtp.gmail.com at your service`¹:

```
Start Time: 1679236390
Timeout   : 7200 (sec)
Verify return code: 0 (ok)
Extended master secret: no
Max Early Data: 0
--
read R BLOCK
250 smtp.gmail.com at your service
```

Далее сервер SMTP запрашивает аутентификацию с помощью команды `AUTH LOGIN`. Мы не будем вдаваться в дальнейшие подробности, но главное здесь то, что команды протокола SMTP передаются между клиентом и сервером в простом текстовом формате. Очень важно защищать канал связи SMTP с помощью шифрования TLS. То же самое относится и к другим почтовым протоколам (POP3 и IMAP).

Итак, мы рассмотрели несколько сетевых служб, в том числе такие, которые могут охватывать много сетей или даже весь интернет. Сетевые пакеты переносят данные и адреса назначения в полезной нагрузке, но не менее важны сигналы синхронизации между оконечными точками соединения — в основном для того, чтобы различать процедуры передачи и получения. Синхронизация сетевых пакетов основана на временных метках. Если бы нельзя было с высокой точностью синхронизировать время между сетевыми узлами, надежный обмен данными по сети был бы невозможен. Далее мы рассмотрим сетевые службы времени.

Серверы NTP

NTP — это стандартный сетевой протокол для синхронизации часов между компьютерами в сети. NTP обеспечивает, чтобы системные часы на тех или иных компьютерах были синхронизированы относительно времени UTC (мирового эталона времени) с погрешностью не больше нескольких миллисекунд.

¹ Код ответа 250; сервер `smtp.gmail.com` к вашим услугам. — *Примеч. пер.*

Протокол NTP обычно реализуется по модели «клиент — сервер». Сервер NTP выступает как источник точного времени, транслируя или отправляя клиентам дейтаграммы с обновленными временными метками. Сервер постоянно корректирует собственные системные часы в соответствии с общеизвестными серверами точного времени по всему миру, преодолевая сетевые задержки с помощью специализированных алгоритмов.

Проверить состояние синхронизации NTP в Linux можно довольно простым способом — утилитой `ntpstat`. В Ubuntu ее можно установить с помощью команды `sudo apt install ntpstat`, а в Fedora — с помощью команды `sudo dnf install ntpstat`.

Прежде чем использовать `ntpstat`, нам понадобится локально запущенный сервер NTP. Вот как его можно настроить в Ubuntu (здесь мы используем Ubuntu 22.04.2 LTS):

- Установите пакет `ntp` с помощью такой команды:

```
sudo apt install ntp
```

- Проверьте состояние службы `ntp`:

```
sudo systemctl status ntp
```

- Включите службу `ntp`:

```
sudo systemctl enable ntp
```

- Измените настройки межсетевого экрана:

```
sudo ufw allow from any to any port 123 proto udp
```

- Установите пакет `ntpdate`:

```
sudo apt install ntpdate
```

- Перезапустите службу `ntp`:

```
sudo systemctl restart ntp
```

Прежде чем устанавливать `ntp`, учтите, что в Ubuntu по умолчанию вместо `ntpd` используется другой демон, который называется `timesyncd`. Если установить `ntpd`, стандартный демон будет отключен.

Чтобы запросить состояние синхронизации NTP, можно запустить команду `ntpstat`:

```
packt@neptune:~$ ntpstat
synchronised to NTP server (31.209.85.242) at stratum 2
time correct to within 29 ms
polling server every 64 s
```

Команда `ntpstat` предоставляет IP-адрес сервера NTP, с которым синхронизируется система (в данном случае `31.209.85.242`), погрешность синхронизации (29 миллисекунд) и интервал времени, с которым опрашивается сервер для

повторной синхронизации (64 секунды). Чтобы больше узнать об этом сервере, можно запросить его IP-адрес с помощью такой команды:

```
dig -x 31.209.85.242
```

Похоже, что это один из серверов времени компании LWLcom (ntp1.lwlcom.net):

```
packt@neptune:~$ dig -x 31.209.85.242

; <<>> DiG 9.18.30-0ubuntu0.24.04.2-Ubuntu <<>> -x 31.209.85.242
;; global options: +cmd
;; Got answer:
;; ->HEADER<<- opcode: QUERY, status: NOERROR, id: 8766
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;242.85.209.31.in-addr.arpa.      IN      PTR

;; ANSWER SECTION:
242.85.209.31.in-addr.arpa. 1999 IN      PTR      ntp1.lwlcom.net.

;; Query time: 30 msec
;; SERVER: 10.255.255.254#53(10.255.255.254) (UDP)
;; WHEN: Thu Feb 27 04:30:43 MSK 2025
;; MSG SIZE rcvd: 84
```

Соединение между клиентом и сервером NTP использует в качестве транспортного протокола UDP на порте 123. В главе 9 «Как обеспечивать безопасность в Linux» есть раздел, посвященный установке и настройке сервера NTP. Чтобы подробнее узнать об этом протоколе, можно обратиться к странице en.wikipedia.org/wiki/Network_Time_Protocol.

На этом наше краткое путешествие по сетевым серверам и протоколам подошло к концу. Повседневные задачи администрирования Linux часто требуют удаленного доступа к системе. Удаленно обращаться к компьютерам и управлять ими можно разными способами. В следующем разделе описаны некоторые распространенные средства удаленного доступа и соответствующие сетевые протоколы.

Удаленный доступ

Большинство сетевых служб Linux предоставляют довольно ограниченный интерфейс удаленного управления, а их утилиты с интерфейсом командной строки обычно работают локально на той же системе, где запущена служба. В результате для соответствующих административных задач нужен локальный доступ к терминалу, но прямой доступ к системной консоли не всегда возможен. В этом случае на помощь приходят серверы удаленного доступа, которые по-

зволяют организовать виртуальный сеанс входа в терминал на удаленной машине.

Давайте рассмотрим некоторые из наиболее распространенных служб и приложений для удаленного доступа.

SSH

SSH — это, пожалуй, самый популярный протокол защищенного входа в систему для удаленного доступа. Он использует надежное шифрование в сочетании с механизмами аутентификации, чтобы установить защищенный канал связи между клиентом и сервером. Сервер SSH относительно легко установить и настроить, и соответствующая процедура описана в разделе «Как настроить сервер SSH» в главе 13 «Как настраивать серверы Linux». По умолчанию для SSH используется сетевой порт 22.

SSH поддерживает следующие типы аутентификации:

- аутентификация с открытым ключом;
- аутентификация по паролю;
- интерактивная аутентификация с помощью клавиатуры.

В следующих подразделах мы вкратце опишем каждый из этих механизмов.

Аутентификация с открытым ключом

Аутентификация с открытым ключом — это, пожалуй, самый распространенный тип аутентификации по протоколу SSH.

В этом разделе мы будем использовать термины «открытый ключ» и «ключ SSH» как взаимозаменяемые, в основном для того, чтобы отразить соответствующую номенклатуру аутентификации SSH в сообществе Linux.

Для аутентификации с помощью ключа SSH используется пара из **открытого ключа** (он же *сертификат*) и **закрытого ключа**. Как правило, эти ключи создаются утилитой `ssh-keygen` с помощью стандартных алгоритмов шифрования, таких как **RSA** или **DSA**.

Аутентификация с открытым ключом SSH поддерживает модели аутентификации **с привязкой к пользователю** или **с привязкой к узлу** — в зависимости от того, кому принадлежат пары ключей. В первом случае у каждого пользователя есть своя пара ключей для доступа по SSH, а во втором для каждого узла используется всего одна пара ключей.

Далее в этом разделе мы объясним и проиллюстрируем обе модели аутентификации с открытым ключом. Основные процессы квити́рования и входа в систему одинаковы для обеих моделей:

- Сначала клиент SSH генерирует защищенную пару ключей и передает свой открытый ключ серверу SSH. Эта операция выполняется однократно, чтобы сделать возможной аутентификацию с открытым ключом.
- Когда клиент инициирует квитиование SSH, сервер запрашивает открытый ключ клиента и сверяет его с допустимыми открытыми ключами. Если обнаружено совпадение, то квитиование проходит успешно, сервер передает клиенту свой открытый ключ, и устанавливается сеанс SSH.
- Дальнейшее взаимодействие между клиентом и сервером происходит по стандартной схеме шифрования/дешифрования. Клиент шифрует данные своим закрытым ключом, а сервер расшифровывает их с помощью открытого ключа клиента. Когда сервер передает данные клиенту, он шифрует их своим закрытым ключом, а клиент расшифровывает их с помощью открытого ключа сервера.

Аутентификация с открытым ключом SSH также известна как **аутентификация без пароля**, и ее часто используют в сценариях автоматизации, где команды выполняются через несколько удаленных соединений SSH без запроса пароля.

Давайте подробнее рассмотрим механизмы аутентификации с открытым ключом с привязкой к пользователю и с привязкой к узлу:

- **Аутентификация с привязкой к пользователю.** Это самый распространенный механизм аутентификации с открытым ключом. В этой модели у каждого пользователя, который подключается к удаленному серверу SSH, есть своя собственная пара ключей. Разным учетным записям на одном узле (или домене) соответствуют разные пары ключей, каждая из которых обеспечивает доступ к удаленному серверу SSH:



Рис. 7.8. Аутентификация с открытым ключом с привязкой к пользователю

- **Аутентификация с привязкой к узлу.** Это еще одна форма аутентификации с открытым ключом, где каждой системе (узлу) соответствует единственная пара ключей, с помощью которой подключаются все пользователи:

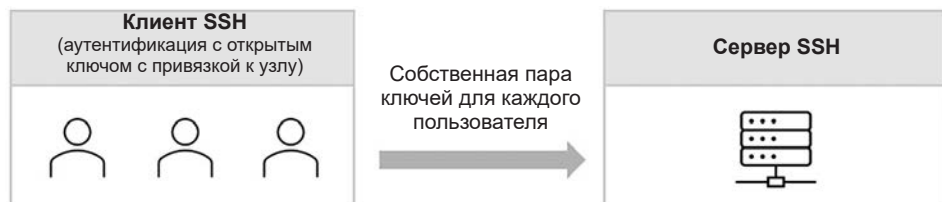


Рис. 7.9. Аутентификация с открытым ключом с привязкой к узлу

При такой аутентификации соответствующая пара ключей SSH может аутентифицировать только те сеансы, которые инициирует один и тот же клиентский узел. Аутентификация с привязкой к узлу позволяет нескольким пользователям подключаться к удаленному серверу SSH с одного и того же узла. Если пользователь попытается использовать ключ, привязанный к одному узлу, на другом узле, доступ будет запрещен.

Иногда механизмы аутентификации с привязкой к пользователю и с привязкой к узлу используются совместно, что обеспечивает повышенный уровень безопасности.

Там, где безопасность не критически важна, могут подойти более простые механизмы аутентификации SSH. Один из них — аутентификация по паролю.

Аутентификация по паролю

При **аутентификации по паролю** клиент SSH предоставляет простой набор учетных данных — таких, как имя пользователя и пароль. Сервер SSH проверяет эти данные, опираясь либо на локальные учетные записи (в файле `/etc/passwd`), либо на избранные учетные записи, которые определены в конфигурации сервера (в файле `/etc/ssh/sshd_config`). Конфигурация сервера SSH подробнее рассматривается в главе 9 «Как обеспечивать безопасность в Linux».

Помимо локальной аутентификации, SSH может использовать такие методы удаленной аутентификации, как Kerberos, LDAP, RADIUS и др. В этих случаях сервер SSH делегирует аутентификацию пользователей удаленному серверу аутентификации, как говорилось в разделе «Серверы аутентификации» в начале этой главы.

Для аутентификации по паролю система должна либо взаимодействовать с пользователем, либо автоматически получать необходимые учетные данные. Еще один подобный механизм — интерактивная аутентификация с помощью клавиатуры.

Интерактивная аутентификация с помощью клавиатуры

Интерактивная аутентификация с помощью клавиатуры предполагает диалог из нескольких последовательных запросов и ответов между клиентом и сервером SSH. Запросы и ответы передаются в формате простого текста, причем сервер может направить пользователю любое количество запросов. В некотором смысле аутентификация по паролю — это частный случай интерактивной аутентификации, когда направляется всего один запрос.

Слово «интерактивный» может создать впечатление, что соответствующая реализация этого метода обязательно должна взаимодействовать с пользователем. На самом деле это не так. Интерактивную аутентификацию с помощью клавиатуры также можно задействовать в реализации механизмов аутентификации на основе специально разработанных протоколов, где соответствующий обмен сообщениями моделируется как протокол аутентификации.

Прежде чем перейти к другим протоколам удаленного доступа, отметим, что протокол SSH широко используется как безопасный, универсальный и эффективный механизм. Однако в некоторых ситуациях подключение по SSH бывает недоступным или недостаточным. Тогда на помощь может прийти *TELNET*, который мы рассмотрим далее.

TELNET

TELNET — это протокол прикладного уровня для двунаправленного обмена данными, где на удаленном узле используется командная строка в режиме простого текста. Исторически TELNET был одним из первых протоколов удаленного соединения, но безопасность по-прежнему остается его слабой стороной. В конце концов SSH стал стандартным способом удаленного входа в систему, однако преимущества TELNET проявляются в ситуациях, когда приходится устранять неполадки в различных протоколах прикладного уровня — например, при соединении с веб-сервером или почтовым сервером. Подробнее о том, как использовать TELNET, вы узнаете в главе 9 «Как обеспечивать безопасность в Linux».

TELNET и SSH — это интерфейсы удаленного доступа, которые управляются через командную строку. Но бывают случаи, когда нужно подключиться к рабочему столу удаленной машины через графический интерфейс. Далее мы рассмотрим совместный доступ к рабочему столу.

VNC

VNC — это система совместного доступа к рабочему столу, которая позволяет пользователям получать доступ к графическому интерфейсу удаленного компьютера и управлять им. Реализации VNC существуют в виде клиент-серверных приложений для разнородных платформ. Например, сервер VNC, который запущен на компьютере под управлением Linux, позволяет получить доступ к рабочему столу многочисленным клиентам VNC, которые работают на Windows

или macOS. Для сетевого обмена данными VNC использует протокол **RFB**, который определен в *RFC 6143*. Настроить сервер VNC довольно просто, и для него нужна настольная система с графическим интерфейсом. Мы расскажем об этом подробнее в главе 13 «Как настраивать серверы Linux».

На этом мы завершаем раздел о сетевых службах и протоколах. Мы постарались охватить наиболее общие понятия, касающиеся сетевых серверов и приложений, которые в основном работают по клиент-серверному или распределенному принципу. Для разных типов серверов мы описали соответствующие сетевые протоколы и поговорили о некоторых особенностях их работы. В главе 9 «Как обеспечивать безопасность Linux» и в главе 13 «Как настраивать серверы Linux» мы представим практические реализации некоторых из сетевых серверов.

В следующем разделе мы обратимся к ключевым задачам сетевой безопасности.

Задачи сетевой безопасности

К **сетевой безопасности** относятся процессы, действия и политики, направленные на то, чтобы предотвращать несанкционированный доступ к компьютерным сетям. Сетевая безопасность охватывает широкий спектр технологий, инструментов и практических приемов. Вот некоторые из них:

- **Управление доступом.** Система выборочно ограничивает доступ на основе аутентификации и авторизации пользователей. К механизмам управления доступом относятся, например, пользователи, группы и права доступа в Linux. Некоторые понятия из этой области рассматривались в главе 4 «Как управлять пользователями и группами».
- **Безопасность приложений.** Система обеспечивает безопасность серверных и прикладных приложений (например, почтовых программ, браузеров и мобильных приложений) и защищает их от угроз. К механизмам безопасности приложений относятся, например, **SELinux**, устойчиво зашифрованные соединения, антивирусы и системы защиты от вредоносного ПО. Мы рассмотрим **SELinux** в главе 10 «Аварийное восстановление, диагностика и устранение неполадок».
- **Безопасность оконечных точек.** Система обеспечивает безопасность как серверов, так и пользовательских устройств (смартфонов, ноутбуков и настольных ПК) в сети. Для этого применяются, например, межсетевые экраны и различные механизмы обнаружения вторжений. Мы рассмотрим межсетевые экраны в главе 10 «Аварийное восстановление, диагностика и устранение неполадок».
- **Сегментация сети.** Компьютерные сети разбиваются на более мелкие сегменты — **виртуальные локальные сети (VLAN)**. Их не следует путать с подсетями, которые отражают логическое разбиение сетей с помощью адресации.

- **VPN.** Доступ к корпоративным сетям из общедоступных сетей или интернета с помощью безопасного зашифрованного туннеля. Мы подробнее рассмотрим VPN в главе 9 «Как обеспечивать безопасность Linux» и в главе 13 «Как настраивать серверы Linux».

В повседневных задачах администрирования Linux всегда имеет смысл настраивать периметр безопасности в соответствии с перечисленными механизмами, причем примерно в том же порядке. Безопасность сети стоит обеспечивать по принципу «изнутри наружу», от управления доступом к VPN, начиная с локальных систем и сетей и заканчивая межсетевыми экранами, VLAN и VPN.

Итоги

В этой главе мы кратко рассмотрели основные принципы того, как работает сетевое окружение Linux. Мы узнали об уровнях и протоколах сетевого взаимодействия, схемах адресации IP, конфигурациях TCP/IP, общеизвестных портах сетевых приложений и даже о VPN. Если вы владеете общими понятиями сетевых технологий, то сможете полнее представлять себе, как устроены конкретные распределенные системы и как оконечные точки приложений обмениваются данными друг с другом.

Некоторые из теоретических аспектов, представленных в этой главе, будут рассматриваться на практике в главе 13 «Как настраивать серверы Linux», где мы сосредоточимся на реальных реализациях сетевых серверов. В главе 10 «Аварийное восстановление, диагностика и устранение неполадок» мы продолжим разговор о том, как устроены механизмы сетевой безопасности и как настраивать межсетевой экран в Linux. Все, что вы узнали до сих пор, поможет вам эффективнее осваивать дальнейший материал книги.

Следующая глава познакомит вас с тем, как писать сценарии для командной оболочки Linux. Вы узнаете о наиболее распространенных возможностях оболочки и о том, как использовать операторы принятия решений, циклы, переменные, массивы и функции.

Вопросы

Вот краткий тест, который поможет вам подытожить и закрепить некоторые основные понятия этой главы:

1. Как модель OSI соотносится с моделью TCP/IP?

Подсказка: вам может помочь рис. 7.3.

2. Назовите хотя бы два протокола из стека TCP/IP и расскажите, где и как они применяются в практических задачах сетевого администрирования или в приложениях, с которыми вы знакомы.

3. На каком уровне сетевой модели работает протокол HTTP? А протокол DNS?

Подсказка: оба протокола работают на одном и том же уровне.

4. Какой класс сети соответствует IP-адресу 192.168.0.1?

Подсказка: обратитесь к рис. 7.5.

5. Какой префикс сети соответствует маске 255.255.0.0?

Подсказка: снова обратитесь к рис. 7.5.

6. Как настроить статический IP-адрес с помощью утилиты `nmcli`?

Подсказка: используйте подкоманду `connection modify`.

7. Как изменить имя узла Linux?

Подсказка: воспользуйтесь утилитой `hostnamectl`.

8. В чем разница между почтовыми протоколами POP3 и IMAP?

9. Чем аутентификация SSH с привязкой к узлу отличается от аутентификации с привязкой к пользователю?

10. В чем разница между SSH и TELNET?

Дополнительная литература

Чтобы получить дополнительную информацию о том, что рассматривалось в этой главе, обратитесь к изданию:

- Роб Ванденбринк (Rob VanderBrink). «Linux for Networking Professionals»¹.

¹ Ванденбринк Р. «Linux для сетевых инженеров». СПб., изд-во «Питер».

8

Сценарии командной оболочки в Linux

Любому современному специалисту по Linux важно не только свободно ориентироваться в командной строке, но и владеть основами программирования для командной оболочки Linux.

В этой главе вы научитесь автоматизировать различные административные задачи, создавая программы для оболочки Linux. Вы узнаете о том, как устроены типичные сценарии оболочки, и о том, как они запускаются и выполняются. При этом мы задействуем большинство команд, с которыми вы уже встречались в предыдущих главах, — в первую очередь это команды для работы с файлами и каталогами, а также для перенаправления ввода и вывода. Попутно вы изучите основы написания сценариев, познакомитесь с особенностями и сложностями программирования для оболочки, а также овладеете такими специализированными инструментами, как `sed` и `gawk`. Мы надеемся, что к концу этой главы вы привыкнете использовать сценарии в повседневной работе и будете готовы самостоятельно оттачивать свое мастерство.

Мы рассмотрим такие темы:

- Основные возможности командной оболочки.
- Как устроен сценарий оболочки.
- Принятие решений, циклы, переменные, массивы и функции.
- Как использовать `sed` и `(g)awk`.

Технические требования

Для упражнений из этой главы нужно, чтобы у вас был один из распространенных дистрибутивов Linux, который работает на сервере, настольном компьютере или в виртуальной машине. В наших примерах в основном используются Ubuntu/Debian и RHEL/Fedora, но все команды, которые мы рассмотрим, в равной степени подходят и для других дистрибутивов Linux, например openSUSE/SLE.

Командная оболочка Linux

В главе 2 «Командная оболочка и файловая система Linux» вы познакомились с оболочкой, изучили доступные виртуальные консоли, типы команд и файловую систему. Все это подготовило необходимую почву для того, чем мы собираемся заниматься в этой главе. Благодаря всему, о чем шла речь ранее в этой книге, вы уже научились уверенно работать с командной строкой и знаете немало распространенных и полезных команд Linux, которые позволяют работать с файлами, управлять пакетами, пользователями и дисками, а также администрировать сетевое окружение. Все эти знания в конечном итоге пригодятся в этой главе, где мы изучим расширенные возможности командной строки, переменные командной строки, регулярные выражения и другие мощные возможности программирования и автоматизации в командной строке Bash.

В следующем разделе мы начнем знакомиться с расширенными возможностями командной оболочки.

Возможности командной оболочки Bash

Оболочка не только выполняет команды, но и обладает многими другими возможностями, которые существенно упрощают работу системного администратора. В частности, в командной строке можно использовать подстановочные знаки и метасимволы, раскрытие скобок и переменные. Кроме того, вскоре вы узнаете о PATH и псевдонимах оболочки.

Прежде чем продолжить, давайте немного углубимся в историю стандартов, на которые опирается командная оболочка. В свое время, когда появилась операционная система Unix, возникла необходимость в стандарте, который бы регулировал ее различные варианты. Поэтому институт IEEE создал **POSIX** (Portable Operating System Interface, переносимый интерфейс операционных систем) — семейство различных стандартов, которые должны были обеспечивать совместимость операционных систем. Этим стандартам (с некоторыми оговорками¹) отвечают Unix и Linux, а также macOS (которая основана на ядре Darwin, производном от Unix), AIX, HP-UX и Oracle Solaris. В POSIX входят различные стандарты для API языка C, определения форматов файлов, структуры каталогов и переменных окружения, спецификации региональных настроек и регулярных выражений, а также наборы символов.

¹ На момент подготовки этого издания ни один из ведущих дистрибутивов Linux не совместим полностью со стандартами POSIX, однако большинство дистрибутивов считаются «по большей части совместимыми» (mostly POSIX-compliant): это означает, что для большинства практических задач их можно считать совместимыми. macOS, начиная с версии 10.5 (2007 год), сертифицирована на совместимость со стандартом POSIX.1-2001, но не с более поздними редакциями. Ознакомиться с POSIX и родственными стандартами можно на странице publications.opengroup.org/standards/unix. — Примеч. ред.

После этого небольшого урока истории давайте вернемся в наши дни и посмотрим, как использовать подстановочные знаки и метасимволы оболочки.

Подстановочные знаки и метасимволы

Чтобы сопоставлять имена файлов, в Linux используются **подстановочные знаки** двух типов:

- **Звездочка**, или **астериск** (*), — соответствует любой строке, состоящей из нуля или более символов.
- **Вопросительный знак** (?) — соответствует одному любому символу.

Чтобы проверить соответствие любому из нескольких символов, их можно заключить в квадратные скобки ([]).

Подстановочные знаки и скобки относятся к **метасимволам** — специальным знакам, у которых есть особое назначение в командной оболочке Linux и других системах семейства Unix. Мы будем иметь дело с такими метасимволами и их сочетаниями:

*	Подстановочный знак (0 или более любых символов)
?	Подстановочный знак (1 любой символ)
[]	Группировка символов для подстановки
\$	Подстановка переменных
\$(команда)	Подстановка команд
`команда`	
\	Экранирование символов
#	Комментарий
	Конвейерная обработка
&	Запуск команды в фоновом режиме
&&	Оператор И
	Оператор ИЛИ
;	Последовательное выполнение команд
()	Группировка команд при последовательном выполнении
<	Перенаправление ввода
<<	Перенаправление ввода с дополнением
>	Перенаправление вывода
>>	Перенаправление вывода с дополнением

Рис. 8.1. Список метасимволов

Давайте рассмотрим два примера с подстановкой команд с помощью метасимволов. Если нужно использовать вывод одной команды внутри другой, это можно сделать любым из двух способов:

```
packt@neptune:~$ echo "Today is `date`"  
Today is Mon Feb 27 02:09:18 PM UTC 2023
```

```
packt@neptune:~$ echo "Today is $(date)"  
Today is Mon Feb 27 02:09:18 PM UTC 2023
```

Целью этого примера было продемонстрировать, как работает подстановка команд в оболочке, но нелишним будет объяснить, что делают эти команды. Мы использовали вывод команды `date` внутри вывода команды `echo`. Команда `echo` — одна из самых простых в Linux: она всего лишь выводит свой аргумент в стандартный вывод. В нашем случае в составе этого аргумента находится команда `date`, которая выводит текущую дату в системе (в данном случае — в формате по умолчанию).

Две или более команд можно объединить с помощью конвейера, который обозначается вертикальной чертой (`|`). Конвейер перенаправляет вывод первой команды на вход второй команды, а если конвейеризировать три или более команд, то вывод второй команды поступит на вход третьей, вывод третьей — на вход четвертой и т. д.

В следующем примере мы запускаем команду `ls -l /etc`, чтобы получить содержимое каталога `/etc` в расширенном формате, и передаем это содержимое по конвейеру команде `less`:

```
ls -l /etc | less
```

Команда `less` выводит данные постранично, позволяя прокручивать содержимое. Чтобы перемещаться по списку, можно нажимать клавиши со стрелками или клавиши `Page Up` и `Page Down`.

Конвейеры и подстановка команд очень полезны, особенно когда приходится работать со сложными командами или писать сценарии, которыми вы займетесь позже в этой главе.

Теперь давайте рассмотрим последовательность из нескольких команд. Сначала мы запустим ее в терминале, а потом добавим метасимволы, чтобы сгруппировать команды и перенаправить вывод в файл:

```
packt@neptune:~$ who; pwd  
packt pts/0 2023-02-27 13:49 (192.168.124.1)  
/home/packt
```

```
packt@neptune:~$ (who; pwd) > users
```

```
packt@neptune:~$ cat users  
packt pts/0 2023-02-27 13:49 (192.168.124.1)  
/home/packt
```

Как видите, две команды, которые мы запустили в первой строке, можно легко сгруппировать с помощью скобок и перенаправить вывод в файл.

Мы использовали метасимволы трех типов: символ *последовательного выполнения команд* (;), скобки для *группировки команд* в последовательности и символ *перенаправления вывода* (>) в файл. Изначально этого файла не было, мы создали его при выполнении команды.

Первая команда в последовательности — `who`, которая выводит в стандартный поток вывода информацию о вошедших в систему пользователях, а вторая — `pwd`, которая выводит путь к текущему рабочему каталогу. В конце мы запустили команду `cat`, которая выводит на экран содержимое нового файла `users`. В следующем разделе мы покажем, как использовать раскрытие скобок в оболочке.

Раскрытие скобок

С помощью фигурных скобок можно «разворачивать» аргументы команды. В отличие от подстановочных знаков, эти скобки применяются не только к именам файлов, но и к любым строкам. В скобки можно заключить одну строку, несколько строк через запятую или последовательность команд.

Далее мы продемонстрируем несколько примеров того, как используется раскрытие скобок. Сначала мы с его помощью удалим сразу два файла из одного каталога, а затем создадим сразу несколько новых файлов одной командой. Допустим, в текущем рабочем каталоге есть два файла, `report` и `new-report`, и мы хотим удалить их оба сразу. Для этого можно запустить такую команду:

```
rm {report,new-report}
```

А теперь давайте создадим пять файлов, имена которых начинаются одинаково: `file1`, `file2`, ..., `file5`:

```
touch file{1..5}
```

Вот результаты этих команд:

```
packt@neptune:~$ ls  
new-report report users
```

```
packt@neptune:~$ rm {report,new-report}
```

```
packt@neptune:~$ ls  
users
```

```
packt@neptune:~$ touch file {1..5}
```

```
packt@neptune:~$ ls  
file1 file2 file3 file4 file5 users
```

Раскрытие скобок — это мощный инструмент, который добавляет гибкости и эффективности в рабочие процессы любого системного администратора. В частности, этот механизм весьма пригодится, когда вы будете учиться писать сценарии.

После того как мы создали эти пять файлов, вы без труда догадаетесь, как с помощью раскрытия скобок удалить несколько файлов одновременно. Введите в консоль следующую команду и посмотрите, что произойдет:

```
rm file{1..5}
```

Убедитесь, что эта команда удалит все пять файлов, которые мы создали ранее. Просмотрите содержимое текущего каталога с помощью `ls`.

В следующем разделе мы поговорим о псевдонимах команд оболочки и узнаем, что они собой представляют и как ими пользоваться.

Псевдонимы команд оболочки

Командная оболочка Linux поддерживает **псевдонимы** — это очень удобный способ заменять длинные команды на короткие альтернативы. Например, в Ubuntu есть заранее определенный псевдоним `ll`, который является сокращением для `ls -a1F`. Вы можете создавать и свои собственные псевдонимы, которые можно сделать временными или постоянными, подобно переменным в программировании. В следующем примере мы изменили команду, которой соответствует псевдоним `ll`:

```
packt@neptune:~$ alias ll
alias ll='ls -a1F'
```

```
packt@neptune:~$ alias ll='ls -l'
```

```
packt@neptune:~$ alias ll
alias ll='ls -l'
```

Это временная модификация: если перезагрузить компьютер или перезапустить оболочку, она вернется к настройке по умолчанию. Чтобы создать постоянный псевдоним, нужно отредактировать файл `~/.bashrc`. Откройте его в своем любимом текстовом редакторе и добавьте строку, с помощью которой мы обновили псевдоним в терминале. Сохраните файл и запустите такую команду:

```
source ~/.bashrc
```

Возможно, даже лучше будет добавить эту строку в новый файл под названием `.bash_aliases`. Чтобы ознакомиться с примерами того, как создавать псевдонимы, просмотрите содержимое стандартного файла `.bashrc`.

`.bashrc` — это скрытый файл сценария, в котором собраны различные настройки сеанса терминала. Кроме того, этот файл может содержать различные функции, которые помогают справляться с повторяющимися задачами. Код из этого файла автоматически выполняется, когда пользователь входит в систему, но его также можно запустить вручную командой `source .bashrc`.

В следующем разделе мы расскажем, что такое переменные оболочки и как их использовать.

Переменные оболочки Bash

В оболочке Bash используются различные типы переменных, как и в любой другой среде программирования. Bash предлагает набор встроенных переменных и механизм косвенной адресации переменных, а также позволяет определять собственные переменные.

Переменные окружения в Linux бывают двух основных типов: **глобальные** и **локальные**. Как правило, их набор одинаков во всех дистрибутивах Linux, за немногими исключениями. Чтобы узнать об особенностях переменных окружения в вашем дистрибутиве, обратитесь к его документации.

Мы расскажем о том, какие переменные особенно широко используются в Linux, начиная со встроенных переменных.

Встроенные переменные оболочки

Вот краткий список некоторых стандартных встроенных переменных:

HOME	Домашний каталог пользователя (например, <code>/home/packt</code>)
LOGNAME USER	Имя пользователя для входа в систему (например, <code>packt</code>)
PWD	Текущий рабочий каталог
OLDPWD	Предыдущий рабочий каталог
PATH	Путь поиска оболочки (список каталогов, разделенных двоеточиями)
SHELL	Путь к текущей оболочке
TERM	Имя терминала в базе данных <code>terminfo</code>

Чтобы вызвать переменную в оболочке, достаточно поставить перед ее именем знак доллара (`$`). Вот небольшой пример, который показывает, как использовать переменные, которые мы перечислили:

```
packt@neptune:~$ echo $HOME
/home/packt
```

```
packt@neptune:~$ echo $LOGNAME
packt
packt@neptune:~$ echo $USER
packt

packt@neptune:~$ echo $PWD
/home/packt

packt@neptune:~$ echo $OLDPWD

packt@neptune:~$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/
local/games:/snap/bin

packt@neptune:~$ echo $SHELL
/bin/bash

packt@neptune:~$ echo $TERM
xterm-256color
```

Следующий пример показывает, как назначать собственные переменные оболочки. Здесь мы присваиваем новой переменной **MYVAR** строковое значение **sysadmin**, а затем выводим ее в стандартный вывод:

```
MYVAR=sysadmin; echo $MYVAR
```

В начале этого раздела перечислены далеко не все переменные, которые доступны по умолчанию в оболочке. Чтобы увидеть все переменные оболочки, запустите команду **printenv**. Если список слишком длинный, его можно перенаправить в файл. В следующем примере список переменных сохраняется в файле **shell_variables**, и его можно просмотреть с помощью **cat** или текстового редактора, например Vim:

```
printenv > ~/shell_variables
```

Здесь тильда (~) обозначает домашний каталог текущего пользователя. Переменные оболочки доступны только внутри оболочки. Если вы хотите, чтобы те или иные переменные были доступны другим программам, которые запускает оболочка, их нужно экспортировать с помощью команды **export**. Если переменная экспортирована из оболочки, она называется **переменной среды**.

Путь поиска оболочки

Переменная **PATH** — одна из важнейших в Linux: она помогает оболочке узнать, где находятся все программы. Когда вы вводите команду в командной строке, система прежде всего ищет соответствующий исполняемый файл в пути поиска. В переменной **PATH** по умолчанию перечислено несколько каталогов, но к ним можно добавить новые — временно или навсегда, в зависимости от того, как вы это сделаете. Чтобы путь к каталогу стал временно доступным для поиска, про-

сто добавьте его в переменную `PATH`. Например, так можно добавить каталог `/home/packt`:

```
packt@neptune:~$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin

packt@neptune:~$ PATH=$PATH:/home/packt

packt@neptune:~$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:/home/packt
```

Чтобы сделать изменения постоянными, нужно отредактировать переменную `PATH` в файле `~/.bash_profile` или `~/.bashrc`.

Некоторые дистрибутивы, такие как openSUSE, добавляют в домашний каталог пользователя дополнительный каталог `bin`. Там можно размещать файлы, которые вы хотите запускать из-под командной оболочки, например файлы сценариев.

Переменная оболочки `PATH` очень важна, особенно если вы работаете со сценариями, потому что их желательно размещать в каталоге, который известен оболочке. В следующем разделе вы научитесь создавать свои первые сценарии Bash.

Введение в сценарии оболочки

Мы уже рассмотрели важные особенности командной строки Linux, переменные оболочки, подстановочные знаки и другие метасимволы. Теперь пора перейти к тому, что такое сценарии, как их создавать и как использовать их в командной строке Linux. Здесь мы обойдемся без графического интерфейса и будем работать только в командной строке, с которой в основном имели дело в предыдущих главах. Начнем с некоторых элементарных, но важных понятий сценариев командной оболочки.

Для начала разберемся, что такое **сценарий** (или **скрипт**). В словарях написано, что сценарий — это последовательность инструкций, которые выполняются компьютером, в основном для того, чтобы автоматизировать ту или иную задачу. Поскольку команды — это тоже инструкции, сценарием можно считать последовательность команд, которые выполняются как единое целое с помощью оболочки. Это упрощенное определение, но оно передает суть понятия. Давайте посмотрим, как создать файл сценария.

Как создать файл сценария оболочки

Самый естественный способ написать сценарий — это создать его в виде файла, который называется **файлом сценария оболочки**. В Linux файлам такого типа часто присваивают расширение `.sh` для наглядности, однако это не обязательно, потому что, в отличие от Windows, в Linux у расширений файлов нет специальной семантики. Отличительным признаком, по которому файл считается сценарием, служит *самая первая строка* кода внутри файла. В случае сценария оболочки Bash эта строка выглядит так:

```
#!/bin/bash
```

Когда система открывает и выполняет файл, первая строка сообщает командному интерпретатору, что он имеет дело с файлом сценария, который предназначен для определенной оболочки — в нашем случае Bash. Чтобы сценарий обрабатывала другая оболочка, путь к ней нужно указать в первой строке вместо `/bin/bash`. Решетка (`#`) внутри сценария оболочки обычно означает начало однострочного комментария, но в самой первой строке решетка в сочетании с восклицательным знаком (`!`) указывает на командную оболочку. Конструкция `#!` также называется **шебанг** (shebang)¹.

Давайте создадим простейший файл сценария. Создайте в домашнем каталоге файл `basic-script.sh` с таким кодом:

```
#!/bin/bash
whoami
who
date
uptime
```

В этом элементарном сценарии мы использовали четыре разные команды. Мы поместили каждую команду на отдельной строке, но могли бы сделать и по-другому: написать их все на одной строке и разделить точками с запятой. Однако для наглядности лучше разбивать команды по строкам.

Теперь, когда у нас есть первый сценарий, давайте запустим его. Мы создали файл сценария в своем домашнем каталоге под именем `basic-script.sh`. Это легко проверить:

```
packt@neptune:~$ ls -la basic-script.sh
-rw-rw-r-- 1 packt packt 35 Mar 26 06:53 basic-script.sh
```

¹ Существуют разные версии происхождения слова «шебанг» (shebang); чаще всего его считают стяжением от `sharp` — `bang` или `hash` — `bang` (`sharp` и `hash` — неформальные названия символа `#`, а `bang` — символа `!`). Также не позже второй половины XIX в. слово `shebang` начало употребляться в США в значениях приют, притон, магазинчик, вертеп и пр., а позже превратилось в просторечное слово с универсальным значением (the whole shebang — вся эта дребедень). — *Примеч. пер.*

Попробуем запустить его, просто указав имя файла. В результате мы увидим сообщение об ошибке:

```
packt@neptune:~$basic-script.sh
basic-script.sh: command not found
```

Возможно, вам интересно, откуда взялась эта ошибка. Причина в том, что оболочка не знает о вашем сценарии; она не находит его в своей переменной `PATH`. Как вы помните из предыдущего раздела, оболочка обращается к `PATH`, чтобы разрешить местоположение файлов, которые она запускает. Справиться с этой ошибкой можно двумя способами:

- добавить в `PATH` каталог, в котором находится наш сценарий;
- указать относительный или абсолютный путь при вызове сценария в командной строке.

Мы будем использовать второй способ, потому что он удобнее. Однако попробуйте в качестве упражнения самостоятельно использовать первый метод и добавить домашний каталог в переменную `PATH`. Но пока давайте вызовем файл сценария, указав его местоположение:

```
$ ./basic-script.sh
```

Мы опять получим ошибку, на этот раз другую: `Permission denied` (отказано в доступе). Это происходит потому, что у нас нет прав доступа на выполнение файла. Когда мы создали файл, Ubuntu присвоила ему права на чтение и запись только для владельца файла и его группы в соответствии со значением `umask` по умолчанию. Чтобы запускать сценарий, нужно сделать его файл исполняемым с помощью такой команды:

```
$ chmod u+x basic-script.sh
```

После того как мы задали права на выполнение файла, его можно запустить снова, и на этот раз сценарий будет выполнен. Можете убедиться, что все команды внутри сценария выполняются:

```
packt@neptune:~$ ./basic-script.sh
packt
packt tty1 2023-03-26 06:44
packt pts/0 2023-03-26 06:45 (192.168.124.1)
Sun Mar 26 07:17:36 AM UTC 2023
07:17:36 up 33 min, 2 users, load average: 0.00, 0.00, 0.00
```

Как показано на предыдущем снимке экрана, сценарий успешно выполнен, и в терминал выводится результат каждой команды. В первой строке мы видим имя пользователя (вывод команды `whoami`), во второй — сведения о пользователях, которые вошли в систему (вывод команды `who`), в третьей — текущую дату и время (вывод команды `date`), а в последней — время непрерывной работы и некоторые другие сведения о системе (вывод команды `uptime`).

Как правило, мы советуем использовать в сценариях *как можно больше комментариев*¹. Считается, что это хорошая практика, которая делает процесс программирования более увлекательным и побуждает больше задумываться о том, что вы пишете. Если вы документируете свой код, то ваши сценарии впоследствии будет легче читать как вам самим, так и всем остальным, кто может с ними столкнуться. В примерах этой книги не так уж много комментариев, но это связано с ограничениями по количеству страниц. Не бойтесь перестараться с комментариями в собственном коде.

Итак, вы знаете, как создать файл сценария оболочки и как его запустить. Теперь давайте перейдем к более сложным темам и узнаем, как использовать переменные в сценариях.

Переменные в сценариях оболочки

Мы упомянули переменные в начале этой главы, а теперь пришло время узнать, как использовать их в сценариях. Для начала вспомним, какие типы переменных бывают в Linux. Во-первых, это **переменные окружения**, которые доступны из оболочки, а во-вторых, **пользовательские переменные**, которые создаются пользователями и не доступны напрямую из оболочки. Переменные обоих типов можно использовать в сценариях. Чтобы просмотреть полный список переменных окружения в вашей системе, запустите команду `printenv` или `set`. Некоторые из часто используемых переменных были перечислены в разделе «Встроенные переменные оболочки» ранее в этой главе.

Соглашения об именах переменных

Переменные окружения в оболочке Linux называются только прописными буквами. Этот принцип не обязательно соблюдать, когда вы создаете свои переменные, но в любом случае стоит придерживаться логичной и последовательной схемы именования. На этот счет не существует единых правил, однако следует учитывать, что имена переменных *чувствительны к регистру*, и их рекомендуется делать не слишком длинными (мы советуем *не более 20 символов*). Присвоить значение переменной можно с помощью знака равенства (=)².

¹ Стандартный однострочный комментарий в Bash начинается с символа решетки (#) и продолжается до конца текущей строки. Комментарий может занимать целую строку или размещаться в конце строки после значащего кода. — *Примеч. ред.*

² Вокруг знака равенства не должно быть пробелов, которые допустимы (и даже рекомендуются) во многих других языках программирования. Команда `price=20` присваивает значение 20 переменной `price`, но команда `price = 20` приведет к ошибке. — *Примеч. ред.*

Если вы планируете использовать в именах своих переменных только прописные буквы, задумайтесь о том, к каким катастрофическим последствиям это может привести: ваши переменные станут легко перепутать с переменными окружения, которые именуются по тому же принципу. Мы советуем использовать в именах переменных только *строчные латинские буквы, цифры и знак подчеркивания*. Как вариант, можно делать прописной *первую букву* каждого слова в имени переменной.

Старайтесь обходиться без слишком длинных имен. Используйте краткие и выразительные имена или аббревиатуры, чтобы ваши сценарии было легче читать и понимать.

Теперь давайте узнаем, как определять и использовать переменные внутри оболочки.

Как определять и использовать переменные

Давайте создадим новый файл `user-script.sh`, который будет выводить избранные сведения о пользователе с помощью переменных окружения. После того как мы создадим файл и наберем в нем нужный код, мы сделаем его исполняемым, а затем запустим. Вот соответствующие команды:

```
packt@neptune:~$ touch user-script.sh

packt@neptune:~$ nano user-script.sh
# Открываем файл в nano, редактируем и сохраняем

packt@neptune:~$ cat user-script.sh
#!/bin/bash
echo "Идентификатор пользователя: $UID"
echo 'Имя пользователя:' $USER
echo "Домашний каталог: $HOME"
echo 'Командная оболочка:' $BASH

packt@neptune:~$ chmod u+x user-script.sh

packt@neptune:~$ ./user-script.sh
Идентификатор пользователя: 1000
Имя пользователя: packt
Домашний каталог: /home/packt
Командная оболочка: /bin/bash
```

В предыдущем листинге мы выводили данные с помощью команды `echo` двумя способами. Когда строки заключены в двойные кавычки, имена переменных можно помещать внутри строк, чтобы в выводе отображались их значения. Однако если использовать переменные внутри одинарных кавычек, то значения переменных не будут передаваться интерпретатору. Здесь мы использовали четыре переменные окружения, которые предоставляют сведения о пользова-

теле, — `UID`, `USER`, `HOME` и `BASH`. Это очень простой и ясный пример того, как переменные оболочки можно применять в сценариях.

Вы также можете использовать свои собственные переменные, а не только те, которые предоставляет оболочка. Одна из полезных особенностей интерпретатора состоит в том, что он умеет автоматически определять, какой тип данных использует переменная. Обратите внимание, что если переменная определена внутри сценария оболочки, она будет активна только до тех пор, пока оболочка работает, а после этого переменная пропадет. Давайте создадим новый сценарий, на сей раз с собственными переменными. Это выглядит так:

```
packt@neptune:~$ nano user-variables.sh

packt@neptune:~$ cat user-variables.sh
#!/bin/bash
price=25
product='Футболка'
echo "$product стоит $price $"

packt@neptune:~$ chmod u+x user-variables.sh

packt@neptune:~$ ./user-variables.sh
Футболка стоит 25 $
```

Здесь мы создали файл `user-variables.sh` и определили две переменные: одну с именем `price` и значением 25, а другую с именем `product` и значением `Футболка`. Внутри команды `echo` эти переменные вызываются так же, как переменные окружения. Теперь, когда вы умеете называть, определять и использовать переменные, давайте перейдем к более сложным темам. В следующем разделе мы покажем, как использовать математические выражения в сценариях оболочки.

Как использовать математические выражения в сценариях оболочки

Язык сценариев командной оболочки — это язык программирования, поэтому в нем есть встроенные функции для работы с числами. В оболочке Bash существует команда `expr`, с помощью которой можно выполнять различные математические (а также некоторые строковые и логические) операции. Чтобы узнать обо всех доступных операциях, обратитесь к справочной странице `expr`:

```
man expr
```

Здесь мы продемонстрируем только избранные операции, которые выполняет команда `expr`. Не забывайте, что некоторые символы, которые она использует, нужно экранировать (с помощью обратной косой черты `\`), иначе оболочка неправильно их интерпретирует. Давайте создадим новый файл сценария `math.sh` и включим в него простую арифметическую операцию. Вот как его можно создать и запустить:

```
packt@neptune:~$ nano math.sh

packt@neptune:~$ cat math.sh
#!/bin/bash
price=2000 # Цена товара
vat=200    # НДС
total=$(expr $price + $vat)
echo "Общая стоимость: $total ₽"

packt@neptune:~$ chmod u+x math.sh

packt@neptune:~$ ./math.sh
Общая стоимость: 2200 ₽
```

Вместо `expr` можно также использовать квадратные скобки, которые позволяют проще записывать математические операции. Давайте изменим предыдущий сценарий таким образом:

```
packt@neptune:~$ cat math.sh

#!/bin/bash
price=2000 # Цена товара
vat=200    # НДС
total=${price + $vat}
echo "Общая стоимость: $total ₽"

packt@neptune:~$ chmod u+x math.sh

packt@neptune:~$ ./math.sh
Общая стоимость: 2200 ₽
```

В предыдущих примерах мы использовали целочисленные значения. Если вы попытаете присвоить переменной `vat` дробное значение (например, `2.50`) и запустить сценарий, это приведет к ошибке. Дело в том, что Bash поддерживает только целочисленные арифметические операции. Это ограничение можно преодолеть несколькими способами, наиболее эффективный из которых — использовать команду `bc` (сокращение от Bash calculator, то есть калькулятор Bash).

Если вы хотите, чтобы командная оболочка полноценно поддерживала арифметику с плавающей точкой, обратите внимание на оболочку **Zsh**. Она установлена по умолчанию в некоторых дистрибутивах Linux (например, Manjaro и Kali Linux), а также в macOS. Если хотите, установите ее в свой дистрибутив.

Давайте посмотрим, как можно использовать команду `bc` в нашем сценарии `math.sh`. Изменим код так:

```
packt@neptune:~$ cat math.sh
#!/bin/bash
price=2000 # Цена товара
vat=2.50   # НДС
total=$(echo "$price + $vat" | bc )
```

```
echo "Общая стоимость: $total ₽"

packt@neptune:~$ chmod u+x math.sh

packt@neptune:~$ ./math.sh
Общая стоимость: 22.5 ₽
```

В предыдущем примере мы присвоили переменной `vat` значение с плавающей точкой и применили команду `bc`, чтобы вычислить итоговую сумму. Когда мы присваиваем значение переменной, эту команду можно использовать так:

```
имя_переменной=$(echo "выражение" | bc)
```

Здесь *выражение* — это арифметическая операция (в нашем случае сложение). Мы перенаправили вывод команды `echo` в команду `bc` и присвоили результат переменной `var`.

Сценарии оболочки Bash не сводятся к математическим операциям или последовательному выполнению команд. Иногда нужно, чтобы оболочка принимала решения в зависимости от того, какие входные данные она получила и какого вывода мы от нее ожидаем. В этом случае в дело вступают специальные управляющие конструкции, которые мы рассмотрим в следующем разделе.

Управляющие конструкции в Bash

В этом разделе мы продемонстрируем, как использовать **условия** и **циклы** в Bash. Они могут оказать неоценимую помощь, когда дело дойдет до сложных сценариев оболочки. Кроме того, здесь вы научитесь работать с массивами и узнаете, как сценарии считывают входные данные, а также как форматируются выходные данные.

Массивы в Bash

В предыдущих разделах вы узнали, как использовать переменные. Теперь настало время перейти на новый уровень и познакомиться с **массивами** — более сложной формой переменных. Допустим, вам нужно работать с несколькими переменными, в которых хранятся похожие значения, например имена файлов. Вместо того чтобы использовать много переменных вроде `file1`, `file2`, `file3` и т. д., можно создать массив, в котором будут храниться все эти имена. Если вы владеете другими языками программирования, то, скорее всего, уже встречались с массивами. Но если вы не знаете других языков — не переживайте, ведь в Bash очень удобно работать с массивами.

Начнем с простого примера. Допустим, в нашем сценарии фигурируют разные имена пользователей. Вместо того чтобы заводить отдельную переменную для каждого имени, можно объявить **индексированный массив**:

```
usernames=("paul" "janet" "mike" "john" "anna" "martha")
```

Элементы внутри массива индексируются начиная с 0. Это важно помнить, чтобы правильно обращаться к его содержимому. Если вам нужен доступ к третьему элементу массива `usernames` (строка `mike`), используйте такой код:

```
echo ${usernames[2]}
```

Эта команда выводит `mike` (без кавычек). Чтобы вывести все элементы массива (через пробел), используйте такой код:

```
echo ${usernames[*]} или echo ${usernames[@]}
```

А эта команда выводит размер массива, то есть количество элементов в нем:

```
echo ${#usernames[@]} or echo ${#usernames[*]}
```

В нашем случае в массиве шесть элементов, поэтому выводится число 6.

Если нужно добавить в массив новое имя пользователя (например, `alex`), это можно сделать разными способами. Чтобы просто поместить его в массив, не указывая конкретной позиции, запустите такую команду:

```
usernames+=("alex")
```

Новое имя пользователя будет добавлено в конец массива.

Можно также заменить значение существующего элемента. Например, так можно заменить второй элемент (`janet`) на `zack`:

```
usernames[1]="zack"
```

В предыдущих примерах мы рассматривали массив, который состоял из строк, но элементами индексированного массива могут быть и целые числа.

Массив также можно создать с помощью встроенной команды `declare`. Чтобы объявить индексированный массив, ее можно вызвать так:

```
declare -a имя_массива
```

С ключом `-A` эта команда создает ассоциативный массив:

```
declare -A имя_массива
```

Ассоциативные массивы состоят из пар элементов вида «ключ — значение». Вот как можно объявить ассоциативный массив:

```
declare -A linux_distros=( [KDE]="openSUSE" [GNOME]="Fedora" [Xfce]="Debian"  
[Cinnamon]="Mint" )
```

Внутри квадратных скобок находятся ключи, по которым можно обращаться к значениям, а внутри двойных кавычек — сами значения. Все значения можно вывести с помощью той же команды, что и для индексированных массивов:

```
echo ${linux_distros[@]}
```


А чтобы вывести все ключи, запустите такую команду:

```
echo ${!linux_distros[@]}
```

Основное различие между индексированными и ассоциативными массивами заключается в том, что в индексированном массиве мы обращаемся к каждому элементу по его индексу, то есть порядковому номеру внутри массива, а в ассоциативном — по текстовому ключу.

Массивы — важные структуры данных в Bash, и мы задействуем их позже в этой главе, когда пойдет речь об операторах цикла. Но сначала давайте узнаем, как заставить сценарий читать входные данные и как форматировать выходные данные. Сначала рассмотрим, как считывать данные из стандартного ввода.

Как считывать входные данные

По умолчанию оболочка считывает входные данные из стандартного ввода — то есть с клавиатуры. Чтобы считать данные, можно использовать команду `read`. Она считывает все входные данные до тех пор, пока не будет введен символ новой строки. Это происходит, когда вы нажимаете клавишу `Enter`.

В команде `read` можно указать одну или несколько переменных. В последнем случае каждой переменной будет присвоено по одному слову из стандартного ввода. Вот пример:

```
packt@neptune:~$ read name
alexandru

packt@neptune:~$ echo $name
alexandru

packt@neptune:~$ read a b c d
Monday

packt@neptune:~$ echo $a
Monday

packt@neptune:~$ echo $b

packt@neptune:~$ read a b c d
Понедельник Вторник Среда Четверг

packt@neptune:~$ echo $b
Вторник

packt@neptune:~$ echo $a $b $c $d
Понедельник Вторник Среда Четверг
```

В предыдущем листинге мы, в частности, использовали команду `read` с четырьмя переменными `a`, `b`, `c` и `d`. Когда потом мы ввели данные в первый раз, то нажали *Enter* после того, как набрали первое слово на клавиатуре. Тем самым мы

задали значение только для первой переменной `a` и не обеспечили никаких значений для остальных переменных. А во второй раз мы ввели значения для всех переменных и нажали *Enter* только после слова *Четверг*. В результате каждая переменная получила свое значение.

У команды `read` есть несколько ключей, но чтобы подробно изучить их, вам придется обратиться к справочной странице. Эта команда может принимать данные не только из стандартного ввода, но и из файла — с помощью перенаправления. Например, если у вас есть файл `week-days`, его содержимое можно перенаправить команде `read`:

```
packt@neptune:~$ cat week-days
Понедельник Вторник Среда Четверг Пятница Суббота Воскресенье

packt@neptune:~$ read d1 d2 d3 d4 d5 d6 d7 < week-days

packt@neptune:~$ echo $d1 $d3 $d5 $d7
Понедельник Среда Пятница Воскресенье
```

Команду `read` используют в сценариях, когда нужно принять входные данные. Мы продемонстрировали, как применять ее в командной строке, но мы еще вернемся к ней позже в этой главе, когда будем рассматривать более сложные сценарии. В следующем разделе вы научитесь форматировать выходные данные.

Как форматировать выходные данные

В Linux стандартный вывод по умолчанию направляется на монитор. Управлять этим выводом позволяют две команды: `echo`, которую мы уже много раз использовали в этой книге, и `printf`, которую мы рассмотрим в этом разделе.

Команда `printf` похожа на одноименную функцию в языке программирования C. Справочная страница этой команды описывает ее синтаксис:

```
printf формат [аргумент] ...
```

Все аргументы команды выводятся в соответствии с заданной спецификацией формата. Строки могут содержать обычные символы или **управляющие последовательности**, каждая из которых состоит из обратной косой черты и буквы. Вот некоторые широко используемые последовательности:

<code>\n</code>	Новая строка
<code>\t</code>	Горизонтальная табуляция
<code>\\</code>	Символ <code>\</code>
<code>%%</code>	Символ <code>%</code>

Полный список управляющих последовательностей можно найти на справочной странице (`man printf`).

Кроме управляющих последовательностей, команда `printf` использует спецификаторы формата. Вот некоторые из них:

Спецификаторы строки	<code>%s</code>	Обычная строка
	<code>%b</code>	Строка с поддержкой управляющих последовательностей
Спецификаторы числа	<code>%d</code>	Целое число
	<code>%f</code>	Число с плавающей точкой

Теперь, когда вы знаете основы команды `printf`, давайте рассмотрим несколько примеров ее использования.

В следующем упражнении демонстрируется, чем различается вывод со спецификаторами формата и без них:

```
packt@neptune:~$ printf "Привет"
Приветpackt@neptune:~$ printf "%s\n" "Привет"
Привет
```

```
packt@neptune:~$ printf "%b\n" "Первая строка" "Вторая строка"
Первая строка
Вторая строка
```

Как видите, по умолчанию `printf` просто выводит текст, который заключен в двойные кавычки, но не переводит строку после этого. Если использовать спецификатор формата `%s`, команда выводит то, что передано в качестве аргументов (в нашем случае текст между кавычками), и интерпретирует это как строковые данные. Если добавить `\n`, то после каждого аргумента будет вставлен перевод строки.

Давайте теперь используем `printf` внутри сценария. В этом примере фигурирует спецификатор формата `%s`. Обратите внимание, что в одном случае спецификация формата заключена в одинарные кавычки, чтобы продемонстрировать, что вывод будет таким же, как и с двойными кавычками:

```
packt@neptune:~$ cat user-data.sh
#!/bin/bash
printf "Ваше имя: "
read FirstName
printf "Ваша фамилия: "
read LastName
printf 'Добро пожаловать, %s\n' "$FirstName $LastName"
```

```
packt@neptune:~$ ./user-data.sh
Ваше имя: Александр
Ваша фамилия: Пушкин
Добро пожаловать, Александр Пушкин
```

В предыдущем листинге показан сценарий, который считывает две переменные со стандартного устройства ввода, а затем выводит их на стандартное устройство

вывода (монитор). Здесь мы использовали спецификатор строки (%s) и последовательность экранирования новой строки (\n).

Теперь давайте углубимся в форматирование вывода. В следующих примерах мы впервые используем управляющую последовательность табуляции (\t) вместе с переходом на новую строку (\n) и спецификатором строки. Взгляните на код и посмотрите, как работает форматирование:

```
packt@neptune:~$ printf "%s\t%s\n" "№" "Продукт" "1" "Камень"
№      Продукт
1      Камень

packt@neptune:~$ printf "%s\t%s\n" "№" "Продукт" "1" "Камень" "2" "Ножницы"
№      Продукт
1      Камень
2      Ножницы
```

В этом примере управляющие последовательности служат для того, чтобы имитировать табличное форматирование на выходе. С этой целью можно использовать еще более сложные спецификаторы. Вот пример табличного вывода, где фигурируют спецификаторы строк (%s), целых чисел (%d) и чисел с плавающей точкой (%f):

```
packt@neptune:~$ cat format-output.sh
#!/bin/bash
separator=====
separator=$separator$separator
header="\n %-13s %16s %15s %11s\n"
format=" %-13s %09d %8s %7.2f\n"
width=36
printf "$header" "ПРОДУКТ" "Артикул" "СТЕЛЛАЖ" "ЦЕНА"
printf "%$width.${width}s\n" "$separator"
printf "$format" \
Камень 2876 D02 10 \
Ножницы 8748 M05 58.75 \
Бумага 3243 C11 25.5
```

```
packt@neptune:~$ ./format-output.sh
```

```
ПРОДУКТ  АРТИКУЛ  СТЕЛЛАЖ  ЦЕНА
=====
Камень   000002876    D02     10.00
Ножницы  000008748    M05     58.75
Бумага   000003243    C11     25.50
```

Давайте разберем этот сценарий. Сначала мы определяем переменную `separator`, которая нужна, чтобы вывести графический разделитель между заголовком и данными. Затем создаются переменные `header` и `format`, каждая из которых использует спецификаторы и управляющие последовательности, чтобы описать нужное форматирование. `header` начинается с новой строки (\n), затем идет спецификатор строки длиной 13 позиций с выравниванием по левому краю (%-13s), затем последовательно 16, 15 и 11 позиций с выравниванием по право-

му краю (%16s, %15s и %11s соответственно) и перевод строки в конце (\n). Переменная `format` позволяет форматировать данные таблицы в виде двух строковых столбцов (%s), одного целого числа (%d) и одного числа с плавающей точкой (%f). Целое число соответствует артикулу товара, а число с плавающей точкой — его цене. Чтобы вывести артикул, мы используем спецификатор формата %09d: это значит, что вывод будет занимать 9 позиций. Символ 0 в спецификаторе указывает на то, что все пустые позиции перед значением будут заполнены нулями. Таким образом, даже если в артикуле меньше девяти цифр, перед ним будет выведено столько нулей, чтобы общая длина значения составила 9 символов. Результат всех этих команд показан в нижней части предыдущего листинга, где отображается таблица с названиями товаров, их артикулами, размещением и ценами.

Итак, `printf` — это весьма универсальный и мощный инструмент, который способен создавать впечатляющие эффекты в ваших сценариях. Теперь, когда вы владеете основными инструментами форматирования входных и выходных данных в сценариях Bash, давайте перейдем к другим полезным структурам, которые играют важную роль в программировании для командной оболочки.

В следующем разделе мы рассмотрим коды завершения.

Коды завершения и проверка условий

Чтобы работать с условиями и циклами, важно понимать, что такое **код завершения** команды. Когда выполняется условное выражение или цикл, обычно запускается специальная команда, которая проверяет то или иное условие и возвращает разные значения в зависимости от того, верно оно или нет. Это значение называется *кодом завершения* команды и хранится в специальном параметре, который называется `$?`. Как правило, если команда выполнена успешно, то значение `$?` равно 0, а если команда завершилась неудачно, то `$?` может принимать любое значение от 1 до 255 — чаще всего 1. Эти значения также называются **кодами возврата**.

Помимо кодов завершения, важную роль в управляющих конструкциях Bash играют **структуры проверки условий**, на которые опираются механизмы условных операторов и циклов. Эти структуры считаются ключевыми словами оболочки и представляются таким образом:

[[]]	Возвращает логическое значение (0 или 1) выражения, которое заключено в скобки; в том числе может оперировать регулярными выражениями
(())	Выполняет арифметические операции
test	Проверяет тип файлов, а также сравнивает значения (например, строки или целые числа)

Синтаксис Bash требует отделять содержимое скобок пробелами с обеих сторон — например, [*операция*].

В условных выражениях используются также другие операторы, которые позволяют сравнивать значения. В частности, следующие операторы сравнивают целые числа:

-eg	Равно
-ne	Не равно
-lt	Меньше
-le	Меньше или равно
-gt	Больше
-ge	Больше или равно

Помимо перечисленных операторов, в современных командных оболочках в условных конструкциях применяются и основные математические операторы.

Кроме того, в сценариях оболочки Linux используются параметры (особые переменные), которые соответствуют аргументам команды. Переменная `$0` представляет команду, которая запустила сценарий, а переменные `$1`, `$2`, `$3` и т. д. — соответственно первый, второй, третий и т. д. аргументы, которые переданы команде.

Существуют также операторы сравнения строк:

<code>=:</code> или <code>==</code>	Проверяет, что строки идентичны
<code>!=</code>	Проверяет, что строки <i>не</i> идентичны
<code>\<</code> и <code>\></code>	Сравнение строк; символы <code><</code> и <code>></code> в этом контексте нужно экранировать с помощью символа <code>\</code>

А вот некоторые операторы, которые проверяют **тип файла** и выглядят как ключи команды:

-f	Обычный файл
-d	Каталог
-h или -L	Символическая ссылка
-e	Файл существует

В условных конструкциях также фигурируют логические операторы:

-a	Логическое И
-o	Логическое ИЛИ

Наконец, оператор `-z` позволяет проверить, что операнд — пустая строка; это полезно, если нужно узнать, поступили ли данные.

Операторы проверки условий сложны, но крайне полезны, поэтому очень важно их изучить.

В следующем разделе мы рассмотрим условные конструкции Bash.

Условные выражения `if`

Как и в любом другом языке программирования, в Bash есть условные выражения, а именно `if-then-fi`, `if-then-else-fi` и *вложенные* `if`, а также логические операторы, такие как `&&` (*И*) и `||` (*ИЛИ*). В этом разделе мы покажем, как их использовать.

Давайте рассмотрим, какие бывают выражения с `if`:

- Наиболее распространенная форма выражения `if-then-fi` выглядит так:

```
if [ условие ]      # если выполняется условие,  
then                # то запустить следующие команды  
    команды  
fi
```

Эта конструкция проверяет условие, которое указано после ключевого слова `if`, то есть выполняет команду в квадратных скобках. Если команда завершилась успешно (то есть возвращает код завершения 0), то запускаются команды, которые перечислены между ключевыми словами `then` и `fi`.

- Форма `if-then-else-fi` похожа на `if-then` и устроена так:

```
if [ условие ]      # если выполняется условие,  
then                # то запустить следующие команды  
    команды  
else                # иначе запустить следующие команды  
    команды  
fi
```

Как и в случае `if-then`, *условие* запускается, после чего сценарий ведет себя по-разному в зависимости от его кода завершения. Если условие завершается успешно, то выполняются команды между `then` и `else`, но если код завершения отличен от нуля, то выполняются команды между `else` и `fi`. Это дает больше простора для маневра в зависимости от того, какое значение возвращает условие. А если в рамках одной команды `if-then` нужно проверить несколько условий, можно использовать вложенные выражения `if`.

Теперь давайте рассмотрим основные условные аргументы, которые используются в выражениях `if`, и приведем несколько примеров.

- В этом примере мы проверим, какое число ввел пользователь — четное или нечетное. Сценарий принимает число от пользователя с помощью команды `read`, а затем проверяет, равен ли нулю остаток от деления этого числа на 2. Таким образом он определяет, является ли число четным. В этом примере мы будем использовать выражение `if-then-else`, а также команды `read` и `printf`. Вот соответствующий код и результат его выполнения:

```
packt@neptune:~$ cat even_odd.sh
#!/bin/bash
echo "Введите число:"
read number
if [ $(( number % 2 )) == 0 ]
then
    printf "%s\n" "Число $number четное."
else
    printf "%s\n" "Число $number нечетное."
fi
```

```
packt@neptune:~$ ./even_odd.sh
Введите число:
10
Число 10 четное.
```

```
packt@neptune:~$ ./even_odd.sh
Введите число:
5
Число 5 нечетное.
```

- Следующий сценарий с помощью команды `test -f` проверяет, ввел ли пользователь путь к существующему файлу. Будем рассматривать абсолютные пути в файловой системе. Сценарий называется `testing_file.sh`, а его код показан на следующем снимке экрана:

```
packt@neptune:~$ cat testing_file.sh

#!/bin/bash
# Проверяем, что введенный путь соответствует существующему файлу

echo "Введите абсолютный путь к файлу:"
read filename

if [[ -z $filename ]]
then
    printf "%s\n" "Не задан путь к файлу"
    exit 1
else
    printf "Вы ввели путь: %s\n" "$filename"
fi

printf "%s\n" "Проверяем, что $filename является файлом..."
```



```
test -f "$filename"
if (( $?==0 ))
then
    printf "%s %d\n" "Путь $filename соответствует файлу (0 = true):" $?
else
    printf "%s %d\n" "Путь $filename не соответствует файлу (1 = false):" $?
fi
```

Если запустить этот сценарий, он предложит вам указать полный путь к файлу (существующему или нет). Давайте запустим его несколько раз и проверим, правильно ли он работает в трех различных случаях: когда мы не указываем имя файла, когда мы вводим правильное имя файла и когда вводим неправильное:

```
packt@neptune:~$ ./testing_file.sh
```

Введите абсолютный путь к файлу:

Не задан путь к файлу

```
packt@neptune:~$ ./testing_file.sh
```

Введите абсолютный путь к файлу:

/etc/passwd

Вы ввели путь: /etc/passwd

Проверяем, что /etc/passwd является файлом...

Путь /etc/passwd соответствует файлу (0 = true): 0

```
packt@neptune:~$ ./testing_file.sh
```

Введите абсолютный путь к файлу:

/home/test

Вы ввели путь: /home/test

Проверяем, что /home/test является файлом...

Путь /home/test не соответствует файлу (1 = false): 1

Теперь, когда мы рассмотрели основы условных выражений с ключевым словом `if`, перейдем к циклам.

Циклы Bash

В основе **циклов** в Bash лежат команды `for`, `while` и `until`, и в этом разделе мы продемонстрируем, как их использовать. Циклы позволяют запрограммировать повторяющиеся операции — например, когда нужно снова и снова выполнять определенную последовательность команд до тех пор, пока не будет выполнено то или иное условие.

Цикл *for*

Как и в большинстве языков программирования, в Bash существуют простые циклы, за которые отвечает команда `for`. Она выглядит так:

```
for переменная in список
do
    команды
done
```

Если вы раньше не сталкивались с такими выражениями, мы поможем вам разобраться, что они означают. *Переменная* пробегает ряд значений, которые составляют *список*, по одному значению за каждую итерацию. В начале первой итерации переменной присваивается начальный элемент списка, а на каждой следующей итерации будет использоваться очередной элемент, пока мы не достигнем последнего элемента. Сколько элементов в списке, столько и будет итераций. На каждой итерации будут выполняться *команды*, которые находятся между *do* и *done*. Так устроен цикл *for* в общем виде.

Давайте посмотрим, как использовать этот цикл на практике, и для этого запустим его по статически объявленному массиву. Это значит, что сценарий не будет принимать данные от пользователя; мы зададим массив непосредственно в коде. Временная переменная *i* (так называемый счетчик) будет пробегать все элементы массива по порядку. Чтобы подставить весь массив в выражение цикла, его нужно записать в форме `${array[@]}`. В следующем листинге показан код сценария и результат, который он выводит. Обратите внимание, что значения в нашем массиве уже упорядочены по возрастанию, так что это не алгоритм сортировки:

```
packt@neptune:~$ cat array_loop_1.sh
#!/bin/bash
array=(0 1 2 3 4 5 6)
# Цикл по данному массиву
for i in ${array[@]}
do
    echo $i
done

packt@neptune:~$ ./array_loop_1.sh
0
1
2
3
4
5
6
```

В следующем примере мы продолжим работать с массивами и используем их внутри цикла *for*, чтобы показать, как сортировать массив. Мы задействуем большинство структур, которые вам уже знакомы, — помимо циклов и массивов, к ним относится чтение входных данных и форматирование выходных данных.

Алгоритмы сортировки выходят за рамки темы этой книги. Мы рассмотрим только один из них (пузырьковую сортировку), чтобы продемонстрировать, как использовать массивы с циклами *for* и насколько мощным может быть Bash. Однако если вы намерены писать по-настоящему сложные сценарии командной оболочки, мы советуем попробовать другой язык программирования, который лучше подходит для этой цели, например Python. Это невероятно универсальный язык, с помощью которого можно успешно решать многие административные задачи.

Давайте вернемся к задаче сортировки. Возьмем неупорядоченный массив целых чисел и попробуем его отсортировать. Чтобы было интереснее, мы попросим пользователя ввести элементы массива вручную. Так выглядит код сценария:

```
packt@neptune:~$ cat array_bubble_sort.sh

#!/bin/bash

# Запрашиваем длину массива
echo "Введите количество чисел в массиве:"
read n

# Запрашиваем элементы массива
echo "Введите числа:"
for (( i = 0; i < $n; i++ ))
do
    read num[$i]
done

# Начинаем сортировку
for (( i = 0; i < $n; i++ ))
do
    for (( j = $i+1; j < $n; j ++ ))
    do
        if [ ${num[$i]} -gt ${num[$j]} ]
        then
            k=${num[$i]}
            num[$i]=${num[$j]}
            num[$j]=$k
        fi
    done
done

# Выводим отсортированный массив
printf "%s\n" "Отсортированный массив:"
for (( i = 0; i < $n; i++ ))
do
    echo ${num[$i]}
done
```

Давайте разберемся, что здесь к чему. Мы задали длину массива с помощью переменной `n`, а затем использовали ее, чтобы перебрать все числа, которые ввел пользователь. В первом цикле `for` счетчик пробегает значения от 0 до `n-1`; значение `n` не включается, потому что условию цикла `i < $n` соответствуют только те индексы, которые строго меньше `n`. На каждом шаге сценарий запрашивает число от пользователя, а затем увеличивает счетчик на 1 с помощью структуры `i++`. Числа, которые ввел пользователь, сохраняются в массиве `num`. Второй цикл сортирует этот массив и состоит из двух вложенных команд `for` и одной `if`. Во внешнем цикле счетчик `i` по-прежнему пробегает все индексы массива от 0 до `n-1`, а во внутреннем используется отдельный счетчик `j`, который перебирает индексы от `i+1` до `n-1`. Оператор `if` сравнивает `i`-й и `j`-й элементы массива и, если первый из них окажется больше второго, меняет местами эти элементы

в массиве. Для этого используется вспомогательная переменная `k`, которая хранит большее значение во время обмена. Цикл завершается, когда все числа будут перебраны. Третий и последний оператор `for` выводит содержимое отсортированного массива.

Здесь показано, что вводит пользователь и что выводит сценарий:

```
packt@neptune:~$ ./array_bubble_sort.sh
```

```
Введите количество чисел в массиве:
```

```
3
```

```
Введите числа:
```

```
45
```

```
24
```

```
56
```

```
Отсортированный массив:
```

```
24
```

```
45
```

```
56
```

В предыдущем примере пузырьковая сортировка работает так:

- Программа сравнивает первые два элемента массива (45 и 24) и убеждается, что 45 больше, чем 24. Программа меняет эти элементы местами, и массив становится таким: (24 45 56).
- Затем программа сравнивает первый и третий элементы массива (24 и 56) и оставляет их на месте, потому что $24 < 56$.
- Наконец, сравниваются второй и третий элементы (45 и 56), которые тоже остаются на месте. Массив отсортирован.

Пузырьковая сортировка — не самый эффективный алгоритм сортировки, но он помог нам проиллюстрировать, как использовать массивы, команды `for` и `if`, ввод данных от пользователя и форматирование вывода. Чтобы подробнее узнать об алгоритмах сортировки, рекомендуем вам обратиться к тематическим ресурсам в интернете или к источникам, которые перечислены в разделе «Дополнительная литература» в конце этой главы.

Теперь, когда вы знаете, как использовать цикл `for`¹, давайте перейдем к циклу `while`.

¹ Цикл с заданным количеством итераций можно инициировать и другими способами. Например, цикл `for` с 10 итерациями можно задать в форме `for i in {1..10} do команды; done` или `for i in $(seq 10) do команды; done`. В оболочке Zsh есть встроенная команда `repeat`, которая позволяет обойтись без переменной-счетчика, например: `repeat 10 echo Привет!`. — *Примеч. ред.*

Цикл *while*

Цикл *while* похож на *for*, с той лишь разницей, что он в некотором роде включает в себя функции команды *if*. Цикл повторяет *команды* до тех пор, пока *условие* истинно:

```
while условие
do
    команды
done
```

Условие проверяется в начале каждой итерации. Если оно истинно (то есть код завершения равен 0), то выполняются указанные команды, после чего цикл заходит на следующую итерацию, и так до тех пор, пока условие не станет ложным.

Давайте рассмотрим примеры. В следующем сценарии используется оператор *while*, чтобы вывести в порядке убывания все числа от того, которое ввел пользователь, до 1:

```
packt@neptune:~$ cat while_loop_1.sh
```

```
#!/bin/bash
# Запрашиваем у пользователя число
printf "Введите целое положительное число: "
read number

printf "Перечисляем числа... \n"

# Начинаем цикл
while [ $number -gt 0 ]
do
    printf " %s " $number
    number=$(( $number - 1 ))
done
printf "%s\n"
```

```
packt@neptune:~$ ./while_loop_1.sh
Введите целое положительное число: 10
Перечисляем числа...
10 9 8 7 6 5 4 3 2 1
```

```
packt@neptune:~$ ./while_loop_1.sh
Введите целое положительное число: 30
Перечисляем числа...
30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11
10 9 8 7 6 5 4 3 2 1
```

Команда *while* проверяет условие `[ $number -gt 0 ]` и выполняет итерации до тех пор, пока оно не станет ложным. Это значит, что команды между *do* и *done* будут повторно выполняться до тех пор, пока *number* не перестанет быть больше 0. Вторая из этих команд на каждой итерации уменьшает число на 1: если этого не делать, то оно никогда не достигнет 0, и вы попадете в бесконечный цикл. Мы

запустили сценарий дважды и сначала ввели число 10, а затем 30; результаты вы видели на предыдущей странице.

`while` — это простой и полезный цикл, который является отличным дополнением к циклу `for`. Теперь давайте рассмотрим цикл `until`.

Цикл `until`

Эта циклическая структура работает как цикл `while` «наоборот». Она зависит от условия, которое с самого начала является ложным, и команды между `do` и `done` повторно выполняются, пока оно не станет истинным. Синтаксис цикла выглядит так:

```
until условие
do
    команды
done
```

В качестве элементарного примера переделаем предыдущий цикл `while` так, чтобы он работал с помощью команды `until`. Вот соответствующий код:

```
#!/bin/bash

# Запрашиваем у пользователя число
printf "Введите целое положительное число: "
read number

printf "Перечисляем числа... \n"

# Начинаем цикл
until [ $number -eq 0 ]
do
    printf " %s " $number
    number=$(( $number - 1 ))
done
printf "%s\n"
```

Вы поняли, в чем разница между циклами `until` и `while`? В случае `until` итерации продолжаются до тех пор, пока значение переменной `number` не станет равным нулю (это проверяет условие `[ $number -eq 0 ]`). Сами команды внутри цикла — те же, что и в случае `while`, но условие выхода отличается. Вывод, как и следовало ожидать, будет таким же, как и при использовании цикла `while`¹.

¹ Вывод будет отличаться, если ввести отрицательное число: в этом случае приведенный здесь сценарий с циклом `until` сначала перечислит в убывающем порядке все целые числа от введенного числа до минимально возможного значения (в случае `5Int64` это $-2^{63} = -9\,223\,372\,036\,854\,775\,808$), а затем все числа от максимально возможного значения ($2^{63} - 1 = 9\,223\,372\,036\,854\,775\,807$) до 1. — *Примеч. ред.*

Прежде чем перейти к более продвинутым структурам программирования, давайте узнаем о том, как управлять выполнением цикла с помощью специальных ключевых слов.

Команды для управления циклом

Две команды, с помощью которых в Bash можно прервать обычное выполнение цикла, — это `break` и `continue`. Они относительно просты и позволяют досрочно завершить весь цикл или его отдельную итерацию. Рассмотрим простой сценарий, который перебирает ряд целых чисел, пока не достигнет заданного значения. Вот его код:

```
#!/bin/bash
echo "Введите целое число (от 0 до 10): "
read limit
echo "Перечисляем числа..."
for (( i = 0; i <= 10; i++ ))
do
    echo $i
    if [ $i -eq $limit ]
    then
        break
    fi
done
```

Сценарий предлагает пользователю ввести число, после чего сохраняет его в переменной `limit` и выводит все целые числа от нуля до `limit`. Цикл `for` перебирает все значения от 0 до 10, пока не достигнет `limit`, после чего цикл прерывается с помощью команды `break`. Проверьте результат, подставляя различные значения. Вот как ведет себя сценарий, если ввести число 5:

```
packt@neptune:~$ ./break_loop.sh
Введите граничное значение:
5
Перечисляем числа...
0
1
2
3
4
5
```

Как видите, цикл завершается после того, как дойдет до значения 5, и выводит его последним в списке чисел. Если нужно, чтобы конечное значение не включалось в вывод, можно отредактировать код, как показано в следующем примере:

```
#!/bin/bash
echo "Введите целое число (от 0 до 10): "
read limit
echo "Перечисляем числа..."
for (( i = 0; i < 10; i++ ))
```

```
do
    if [ $i -eq $limit ]
    then
        break
    fi
    echo $i
done
```

Чтобы значение `limit` не отображалось, мы переместили команду `echo $i` после условной конструкции `if`. В результате сценарий не выводит число, которое задал пользователь, а выводит только все предыдущие числа. Используйте первую или вторую версию сценария в зависимости от того, какова ваша задача.

В следующем примере мы покажем, как использовать команду `continue`, чтобы прервать не весь цикл, а только текущую итерацию. Сценарий будет во многом похож на предыдущий: он будет по-прежнему выводить числа от 0 до 10, но на этот раз не останавливаться на том значении, которое ввел пользователь, а просто пропускать его в выводе. Когда выполняется условие, команда `break` прерывает цикл, а команда `continue` пропускает оставшуюся часть команд между `do` и `done` и переходит на следующую итерацию цикла. Давайте посмотрим код сценария, в котором используется `continue`:

```
#!/bin/bash
echo "Введите число, которое надо пропустить (от 0 до 10): "
read exclusion
echo "Перечисляем числа..."
for (( i = 0; i <= 10; i++ ))
do
    if [ $i -eq $exclusion ]
    then
        continue
    fi
    echo $i
done
```

Как видите, это почти такой же код, как в предыдущем примере, но вместо `break` в нем используется команда `continue`. Посмотрите на результат:

```
packt@neptune:~$ ./continue_loop.sh
Введите число, которое надо пропустить (от 0 до 10):
5
Перечисляем числа...
0
1
2
3
4
6
7
8
9
10
```


Как видите, значение 5 не отображается, потому что команда `continue` работала таким образом, что цикл пропустил соответствующую команду `echo $i`.

К этому моменту вы знаете довольно много о сценариях в Linux. Вы умеете обращаться с переменными, массивами, условными конструкциями и циклами, а также вмешиваться в стандартное выполнение цикла. В следующем разделе мы покажем, как использовать более сложные структуры программирования — такие, как функции.

Как работать с функциями

Как и в большинстве языков программирования, **функции** могут пригодиться, когда вам нужно использовать одни и те же блоки кода в разных местах программы. После того как вы создали функцию, ее можно вызывать в любом месте сценария. Объявить функцию в Bash можно двумя способами. Прежде всего, можно использовать ключевое слово `function`, за которым следует имя функции:

```
function имя_функции {  
    команды  
}
```

Второй способ — задать имя функции и поставить после него круглые скобки:

```
имя_функции() {  
    команды  
}
```

Здесь *имя_функции* — это уникальное имя, по которому можно будет обращаться к функции во всем сценарии. *команды* — это одна или несколько команд оболочки, которые функция выполняет в том же порядке, в каком они перечислены. Говоря простыми словами, функции можно рассматривать как сценарии внутри сценариев. Давайте разберем несколько примеров.

Рискуя утомить вас однообразными примерами, мы тем не менее вернемся к одному из предыдущих сценариев и превратим его в функцию, которая выводит числа от заданного значения до 0. Но сначала — важное замечание о функциях.

Функции нужно создавать до того, как вызывать. Вызов функции — это ее запуск внутри сценария. Если создать функцию после того, как она вызывается, то сценарий сообщит об ошибке, потому что Bash — однопроходной интерпретатор. Хорошим тоном считается объявлять функции в начале сценария — тогда они будут доступны в любой момент, когда понадобятся в сценарии.

Итак, вот наша первая функция. Обратите внимание, что мы сначала создаем функцию `seqn`, а потом вызываем ее:

```

packt@neptune:~$ cat first_function.sh

#!/bin/bash

# Наша первая функция
seqn() {
    printf "Введите целое положительное число: "
    read number
    printf "Перечисляем числа... \n"
    while [ $number -gt 0 ]
    do
        printf " %s " $number
        number=$(( $number - 1 ])
    done
    printf "%s\n"
}

# Выводим сообщение перед тем, как вызывать функцию
echo "Вызываем функцию seqn()..."
seqn
echo "Функция завершила работу"

packt@neptune:~$ ./first_function.sh

Вызываем функцию seqn()...
Введите целое положительное число: 5
Перечисляем числа...
 5  4  3  2  1
Функция завершила работу

```

Теперь вы знаете, как создавать и вызывать функции в сценариях оболочки, однако этим дело не ограничивается. Мы постараемся дать вам достаточно информации о функциях, чтобы вы могли использовать их в сценариях, но если вы хотите узнать больше, обратите внимание на источники, которые перечислены в разделе «Дополнительная литература» в конце этой главы.

В следующем разделе мы рассмотрим расширенные возможности функций, такие как вывод, переменные и работа с массивами.

Расширенные возможности функций

Как мы уже отмечали, каждая функция в оболочке Bash сама по себе работает как сценарий. Это значит, что она может манипулировать переменными, массивами и выводом так же, как целый сценарий. В этом разделе мы поговорим о том, как работать с переменными, массивами и выводом внутри функций.

Переменные в функциях

Внутри функций можно использовать глобальные и локальные переменные. Глобальные переменные доступны в пределах всей системы, а локальные — только внутри функции, в которой они были объявлены. По умолчанию все переменные в Bash определяются как глобальные, в том числе внутри функций.

Чтобы объявить локальную переменную, можно использовать ключевое слово `local`. Давайте на простом примере разберемся, как это работает:

```
packt@neptune:~$ cat variables_1.sh

#!/bin/bash
var1='1'
var2='2'
var_function () {
    local var1='11'
    var2='22'
    echo "Переменные ВНУТРИ функции: var1 = $var1, var2 = $var2"
}
echo "Переменные ПЕРЕД вызовом функции: var1 = $var1, var2 = $var2"
var_function
echo "Переменные ПОСЛЕ вызова функции: var1 = $var1, var2 = $var2"

packt@neptune:~$ ./variables_1.sh
Переменные ПЕРЕД вызовом функции: var1 = 1, var2 = 2
Переменные ВНУТРИ функции: var1 = 11, var2 = 22
Переменные ПОСЛЕ вызова функции: var1 = 1, var2 = 22
```

Обратите внимание, как здесь объявляются переменные. Сначала мы объявили переменные `var1='1'` и `var2='2'`, которые по умолчанию становятся глобальными. Затем мы изменили значения переменных `var1` и `var2` внутри функции `var_function`, но перед первой из них мы поставили ключевое слово `local`, чтобы определить ее как локальную для функции. В результате это *не та же* переменная `var1`, которая была объявлена вне функции. Вторую переменную `var2` мы не стали сопровождать ключевым словом `local`, а использовали так же, как и до функции. Поэтому значения оба раза присваиваются одной и той же переменной `var2`. В результате, когда после функции сценарий выведет значения переменных, только `var2` сохранит новое значение, которое мы присвоили внутри функции, а значение `var1` останется тем же, что в начале, потому что внутри функции одноименная переменная была объявлена локально, и ее значение действовало только в пределах функции.

Массивы в функциях

Работать с массивами в функциях сложнее, чем с переменными. Мы рассмотрим два случая: первый — когда нужно передать массив из сценария в функцию, а второй — когда нужно получить массив из функции. В первом случае нельзя указать имя массива в качестве аргумента функции, потому что при этом функция получит только первый элемент массива. Но можно превратить каждый элемент массива в отдельный аргумент функции, а внутри функции снова упаковать аргументы в массив. Рассмотрим соответствующий код:

```
#!/bin/bash

array_function_1() {
    local array=($@)
```

```

    echo "Массив внутри функции:" ${array[@]}
}

array=(1 3 5 7 11 13 17)
echo "Массив снаружи функции:" ${array[@]}

array_function_1 "${array[@]}"

```

Здесь функция `array_function_1` принимает список аргументов, который составлен из элементов массива `array` благодаря конструкции `"${array[@]}"`. Внутри функции объявляется локальная переменная, которая тоже называется `array` (хотя имена не обязательно должны совпадать). Переменная `$@` содержит список аргументов, которые приняла функция, а скобки `()` позволяют упаковать этот список в массив, который затем присваивается локальной переменной `array`. Можно убедиться, что внутри функции фигурирует тот же самый массив, что и снаружи:

```

packt@neptune:~$ ./arrays_in_functions_1.sh
Массив снаружи функции: 1 3 5 7 11 13 17
Массив внутри функции: 1 3 5 7 11 13 17

```

Во втором случае (когда нужно получить массив из функции) решить задачу напрямую не получится, потому что, в отличие от многих языков программирования, функции в Bash могут возвращать только целочисленный код завершения (например, 0 или 1). Но массив из функции можно передать во внешний мир другими способами. Например, здесь внутри функции `create_array` мы создаем массив из четырех элементов, а затем присваиваем его внешней (глобальной) переменной `new_array`:

```

#!/bin/bash

create_array() {
    local -n array=$1
    array=("Джон" "Пол" "Джордж" "Ринго")
    echo "Массив внутри функции:" ${array[@]}
}

create_array new_array
echo "Массив снаружи функции:" ${new_array[@]}

```

Внутри функции `create_array` создается локальная переменная `array`, которая с помощью именованной ссылки (`-n`) ссылается на первый аргумент функции (`$1`), а этот первый аргумент — имя `new_array`, которое мы передаем, когда вызываем функцию. В результате возникает переменная `new_array`, значением которой является тот же массив, что и внутри функции `create_array`:

```

packt@neptune:~$ ./arrays_in_functions_2.sh
Массив внутри функции: Джон Пол Джордж Ринго
Массив снаружи функции: Джон Пол Джордж Ринго

```

Итак, вы научились создавать сценарии и использовать в них переменные, массивы, управляющие структуры (условия и циклы) и функции. Теперь пора узнать, как применять команды `sed` и `(g)awk` как в командной строке, так и в сценариях.

Команды `sed` и `(g)awk`

И `sed`, и `(g)awk` — это продвинутые инструменты, которые помогают работать с текстовыми файлами. Обе эти команды реализуют одноименные языки программирования¹. Вторая из них также фигурирует под именем `gawk`, потому что так называется реализация языка AWK в рамках проекта GNU, которая предлагает больше возможностей и расширений. Давайте узнаем, как использовать оба этих средства в командной строке.

Как использовать `sed` в командной строке

`sed` — это не просто команда оболочки. Это редактор потоков данных, который преобразовывает файлы на основе строгого набора заранее заданных правил. Команда считывает файл построчно и перерабатывает каждую строку по этим правилам. `sed` — это неинтерактивный потоковый редактор, который вносит изменения на основе программного сценария, поэтому он хорошо подходит для того, чтобы редактировать несколько файлов за один вызов или выполнять рутинные повторяющиеся задачи. Общий синтаксис команды `sed` выглядит так:

```
sed ключи... [сценарий] [файл...]
```

Команда `sed` использует различные сценарные подкоманды, и одна из распространенных подкоманд служит для подстановки текста. Здесь мы рассмотрим только эту подкоманду, но если вам захочется узнать больше о `sed`, то вам поможет множество отличных материалов, которые есть в сети и в печатном виде. Например, обратите внимание на учебное руководство по `sed` на сайте проекта GNU: gnu.org/software/sed/manual/sed.html.

Общий синтаксис для подстановки текста выглядит так:

```
sed 's/регулярное_выражение/замена/флаг'
```

Вот несколько характерных примеров того, как можно использовать `sed`:

- Заменить одно слово на другое в текстовом файле. В этом примере мы используем файл, который называется `роем`, находится в нашем домашнем каталоге и содержит короткое стихотворение. Задача состоит в том, чтобы всюду в файле заменить имя Маша на Глаша. Флаг `g` указывает на то, что опе-

¹ `sed` — сокр. stream editor (потоковый редактор). Название языка AWK образовано по первым буквам фамилий его создателей — Альфреда Ахо (Alfred Aho), Питера Вайнбергера (Peter J. Weinberger) и Брайана Кернигана (Brian Kernighan). — *Примеч. пер.*

рация должна быть глобальной — то есть применяться ко всему текстовому документу. Вот результат:

```
packt@neptune:~$ cat poem
Маша, Маша,
Наша Маша,
С днём рождения, милаша! 🍷
```

```
packt@neptune:~$ sed 's/Маша/Глаша/g' poem
Глаша, Глаша,
Наша Глаша,
С днём рождения, милаша! 🍷
```

Если после этого проверить исходный файл с помощью команды `cat`, можно обнаружить, что `sed` только вывел измененный документ на экран и не внес никаких правок в исходный файл. Чтобы применить изменения к файлу, запускайте `sed` с флагом `-i`.

- В следующем примере мы добавляем отступы величиной в два пробела в начало каждой строки и перенаправляем вывод в новый файл. Мы используем тот же файл `poem`, что и в предыдущем примере. Символ `^` обозначает начало строки:

```
packt@neptune:~$ sed 's/^/  /g' poem > poem_indented
```

```
packt@neptune:~$ cat poem_indented
  Маша, Маша,
  Наша Маша,
  С днём рождения, милаша! 🍷
```

- Теперь попробуем с помощью `sed` вывести только вторую строку файла `poem`, а затем — наоборот, все строки, кроме второй:

```
packt@neptune:~$ sed -n 2p poem
Наша Маша,
```

```
packt@neptune:~$ sed 2d poem
Маша, Маша,
С днём рождения, милаша! 🍷
```

- А вот как вывести с 4-й по 6-ю строки файла — на этот раз возьмем файл `/etc/passwd`:

```
packt@neptune:~$ sed -n 4,6p /etc/passwd
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
```

- Наконец, вот более практическое упражнение. Давайте выведем содержимое файла `/etc/apt/sources.list.d/ubuntu.sources` из Ubuntu без закомментированных строк. Для этого запустите такую команду:

```
sed '/^#/g' /etc/apt/sources.list.d/ubuntu.sources
```

Эта команда находит строки, которые начинаются с решетки (^#) — то есть комментарии, — и заменяет их на пустые строки. Флаг **g** по-прежнему указывает на то, что операция является глобальной для всего файла. Вот часть вывода этой команды; попробуйте испытать ее на своей системе:

```
Types: deb
URIs: http://archive.ubuntu.com/ubuntu/
Suites: noble noble-updates noble-backports
Components: main universe restricted multiverse
Signed-By: /usr/share/keyrings/ubuntu-archive-keyring.gpg
```

В следующем подразделе мы познакомимся с командой **awk**.

Как использовать *awk* в командной строке

AWK — это целый язык сопоставления с шаблонами; полноценный язык программирования, на основе которого был создан Perl. Он позволяет извлекать данные из текстовых файлов и по синтаксису напоминает C. В парадигме AWK файл состоит из записей (строк), а записи — из полей (столбцов). Типичная структура команды **awk** такова:

```
awk 'переменная ~ /шаблон/ {действия}' файл
```

Истинная мощь **awk** выходит за рамки этой главы, поэтому мы покажем всего один простой пример, который может пригодиться будущему системному администратору.

Давайте составим список всех пакетов, которые установлены в Ubuntu. Для каждого пакета мы хотим вывести имя и опустить все остальные подробности. Для этого понадобится такая команда:

```
packt@neptune:~$ sudo dpkg -l | awk '{print $2}' > package-list
```

```
packt@neptune:~$ tail package-list
wireless-regdb
xauth
xdg-user-dirs
xfsprogs
xkb-data
xxd
xz-utils
zerofree
zlib1g:amd64
zstd
```

Обычно для того, чтобы узнать, какие пакеты установлены в Ubuntu, мы запускаем команду **dpkg -l**. В этом примере мы перенаправили результаты этой команды команде **awk**, которая вывела второй столбец (поле) из этих результатов ('{print \$2}'). Полученный вывод мы перенаправили в новый файл с именем **package-list**, а затем запустили команду **tail**, чтобы просмотреть последние 10 строк этого файла.

И `sed`, и `awk` — очень мощные команды, и мы лишь поверхностно коснулись того, на что они способны. Пожалуйста, уделите время тому, чтобы глубже изучить эти два замечательных инструмента.

Демонстрация межпроцессного взаимодействия с помощью сценариев

Мы уже рассматривали **межпроцессное взаимодействие (IPC)** в главе 5. Сейчас мы вернемся к этому механизму, чтобы продемонстрировать соответствующие сценарии командной оболочки.

Наши примеры будут опираться на модель **отправителя и получателя** (они также называются **производителем и потребителем**). У этих сторон есть доступ к общему интерфейсу, куда отправитель записывает те или иные данные и откуда получатель их считывает. Механизмы IPC обычно реализуются в распределенных системах, которые строятся вокруг относительно сложных приложений. В наших примерах мы будем использовать простые сценарии Bash, которые имитируют процессы отправителя (`producer.sh`) и получателя (`consumer.sh`). Надеемся, что, несмотря на простоту, эти модели дадут адекватное представление о том, что происходит в реальных приложениях.

Итак, давайте подробнее рассмотрим такие механизмы межпроцессного взаимодействия, как общее хранилище, именованные и неименованные каналы, а также сокеты, которые мы вкратце представили в главе 5.

Общее хранилище

Самое простое **общее хранилище** IPC может представлять собой обычный файл, который расположен на диске (допустим, файл `storage` в домашнем каталоге). **Отправитель** записывает данные в этот файл, а **получатель** считывает их из того же файла. В любой момент времени потоки данных отправителя и получателя идентичны.

Очевидный недостаток такой простой схемы — целостность операций чтения/записи, между которыми могут возникать условия гонки. Чтобы избежать этой проблемы, файл нужно блокировать во время операций записи, и тогда они не будут перекрываться с другими операциями ввода-вывода. Чтобы не усложнять учебные примеры, здесь мы не будем решать эту проблему, но о ней стоит помнить в среде реальной эксплуатации. В нашем примере отправитель записывает в файл `storage` очередной набор данных (10 случайных значений UUID) каждые 5 секунд. В следующем листинге показан сценарий отправителя `producer.sh`:

```
#!/bin/bash
STORAGE_FILE=~/.storage"
rm -f "${STORAGE_FILE}"
# Удалить предыдущий файл хранилища, если он существовал
```



```
while true # Бесконечный цикл
do
    for (( i=1; i <=10; i++ ))
    do
        uuidgen | tee -a "${STORAGE_FILE}"
        # uuidgen генерирует случайный UUID
        # tee выводит данные в стандартный вывод и файл
        # -a позволяет дописывать в конец файла
    done
    sleep 5s
done
```

Получатель считывает содержимое файла `storage` каждую секунду. В следующем листинге показан соответствующий сценарий `consumer.sh`:

```
#!/bin/bash
STORAGE_FILE="/storage"
while true
do
    while IFS= read -r line; do
        # IFS= запрещает отсекаать пробельные символы в начале и конце строки
        # -r запрещает интерпретировать управляющие последовательности
        echo "${line}"
    done < "${STORAGE_FILE}"
    sleep 1s
done
```

Если запустить оба сценария, то отправитель будет создавать случайные строки и записывать их в файл `storage`, а получатель будет считывать данные из того же файла, например:

```
packt@neptune:~$ ./producer.sh
f2d2cc5f-fe15-4fd4-b918-1efcb893b0b4
16a832f0-5585-48b8-b5e6-744aca491b7d
3e577ea9-bd68-4fa5-b329-7df696434a42
b0e21b56-ada7-4af3-9f43-c37c65f7e4d6
64dfc40a-26db-41ce-8197-2b598cd4fc3e
de1da877-5906-4b72-8211-44fd860ad36c
a7f99d04-076f-43a9-be20-ad5e37ed2c17
7971e697-3e33-4d95-b620-1f2548a2b77f
c0d3b820-18d6-4582-8df3-9f8007292900
166870d0-53d7-4c59-a113-112783bb81a6
^C
packt@neptune:~$ ./consumer.sh
f2d2cc5f-fe15-4fd4-b918-1efcb893b0b4
16a832f0-5585-48b8-b5e6-744aca491b7d
3e577ea9-bd68-4fa5-b329-7df696434a42
b0e21b56-ada7-4af3-9f43-c37c65f7e4d6
64dfc40a-26db-41ce-8197-2b598cd4fc3e
de1da877-5906-4b72-8211-44fd860ad36c
a7f99d04-076f-43a9-be20-ad5e37ed2c17
7971e697-3e33-4d95-b620-1f2548a2b77f
c0d3b820-18d6-4582-8df3-9f8007292900
166870d0-53d7-4c59-a113-112783bb81a6
^C
```

Далее мы покажем, как работают неименованные каналы.

Неименованные каналы

Неименованные каналы, которые также известны как **обычные** каналы, передают выход одного процесса на вход другого. В нашей модели «отправитель — получатель» неименованный канал как механизм IPC между двумя процессами проще всего проиллюстрировать так:

отправитель | получатель

Ключевой элемент этой конструкции — вертикальная черта (`|`). Команда, которая находится слева от черты (отправитель), выводит данные, а команда справа от черты (получатель) их принимает. Чтобы продемонстрировать, как работает неименованный канал, давайте создадим два новых сценария: `producer2.sh` (отправитель) и `consumer2.sh` (получатель):

```
packt@neptune:~$ cat producer2.sh
#!/bin/bash
for (( i=1; i<=10; i++ ))
do
    uuidgen
done

packt@neptune:~$ cat consumer2.sh
echo "Данные на стороне получателя:"
echo "-----"
if [ -t 0 ]
then
    data="$*"
else
    data=$(cat)
fi
echo "${data}"
```

В этой реализации отправитель (`producer2.sh`) выводит в консоль данные (10 случайных строк в формате UUID), а получатель (`consumer2.sh`) считывает и отображает либо данные, которые поступили из стандартного ввода (`/dev/stdin`), либо входные аргументы, если стандартный ввод пуст. Условие `if [ -t 0 ]` проверяет, что входные данные введены с терминала, а не поступили по неименованному каналу.

Вот результат межпроцессного взаимодействия между отправителем и получателем:

```
packt@neptune:~$ ./producer2.sh | ./consumer2.sh
Данные на стороне получателя:
-----
95f57f60-5579-4a5a-8a6e-3db263364b7e
f0ed2a05-4423-44ff-87ed-3429ec96f9f9
ef769457-588e-4160-a0ab-3c96979dd777
0371cb65-861d-4554-bf60-a72235b88012
681753cc-2e29-4526-a52b-d47490a505ac
a1fdbdac-152c-4302-bcdc-2925b6f5467a
3dd4e9a0-5278-4df3-b4dd-65a88cb8f795
6824d6c7-aaf9-4e3d-a99e-e7c773435ca8
8ee0eb99-b80f-4447-a21b-f44de9e0bff3
9b6aa6a1-86e3-4549-849d-e018303a64df
```

Нетрудно убедиться, что это именно те данные, которые выводит процесс-получатель. (Обратите внимание на заголовок "Данные на стороне получателя:" перед строками UUID.)

Один из недостатков неименованных каналов заключается в том, что данные, которые передаются между отправителем и получателем, сами по себе нигде не сохраняются. Если процесс отправителя и/или получателя завершится, то канал исчезнет вместе с данными, которые через него передавались. Эту проблему решают именованные каналы, о которых мы поговорим в следующем разделе.

Именованные каналы

Именованные каналы, также известные как **FIFO**¹, на первый взгляд похожи на неименованные, но существенно отличаются от них по семантике. Неименованный канал сохраняется только до тех пор, пока работают процессы, которые с ним связаны. А именованный канал может сохранять данные и существует до тех пор, пока работает вся система, независимо от состояния процессов, которые к нему подключены.

Обычно именованный канал ведет себя как файл (специального типа, который обозначается буквой *p* в расширенном выводе *ls*), и его можно удалить, когда он больше не нужен. Давайте переделаем наши сценарии отправителя и получателя, чтобы использовать именованный канал как механизм IPC:

```
packt@neptune:~$ cat producer3.sh      packt@neptune:~$ cat consumer3.sh
#!/bin/bash                             #!/bin/bash
PIPE="pipe.fifo"                        PIPE="pipe.fifo"
if [[ ! -p ${PIPE} ]]                   if [[ ! -p ${PIPE} ]]
then                                    then
    mkfifo ${PIPE}                      mkfifo ${PIPE}
fi                                      fi
while true                              while true
do                                      do
    sleep 1; uuidgen | tee "${PIPE}"    if read line <${PIPE}
done                                    then
                                        then
                                        echo "${line}"
                                        fi
done                                    done
```

В этих примерах именованный канал называется `pipe.fifo`. И отправитель, и получатель проверяют, существует ли он, и если нет, то создают его с помощью команды `mkfifo ${PIPE}` (подробнее см. `man mkfifo`). Отправитель каждую секунду записывает в канал случайный UUID (`uuidgen | tee "${PIPE}"`), откуда получатель считывает его при первой возможности (тело цикла `while` в файле `consumer3.sh`).

¹ Сокр. First In, First Out (первым пришел — первым ушел). — *Примеч. пер.*

Если запустить только какой-нибудь один из двух сценариев, то в системе появится соответствующий процесс, но он не будет ничего выводить на экран, а останется в режиме ожидания. Чтобы увидеть, как функционирует именованный канал, нужно инициировать оба сценария — например, запустить их в фоновом режиме с помощью управляющего оператора `&` (в любом порядке):

```
packt@neptune:~$ ./producer3.sh
[1] 1598772
packt@neptune:~$ ./consumer3.sh
[2] 1598803
```

После этого механизм IPC заработает: процесс-отправитель будет каждую секунду отправлять данные в именованный канал `pipe.fifo`, а процесс-получатель — извлекать их оттуда и выводить в терминал:

```
17efa8cf-9e22-45e4-830d-64149c1ece0b
02709804-b90b-43c8-b0d8-3ddb51d1024f
02709804-b90b-43c8-b0d8-3ddb51d1024f
6c8b93ce-154a-4323-90d4-18c0f3fd7a05
6c8b93ce-154a-4323-90d4-18c0f3fd7a05
# и так далее
```

Попробуем остановить один из процессов — например, получателя:

```
packt@neptune:~$ pkill producer3.sh
[2]+  Terminated                  ./consumer3.sh
```

Вы убедитесь, что именованный канал перестал «перекачивать» данные и в консоль снова ничего не выводится. Можно опять запустить `consumer3.sh`, чтобы возобновить межпроцессное взаимодействие:

```
packt@neptune:~$ ./producer3.sh &
[3] 1599096

902c2c11-8ee0-41d8-8c78-1c9fe49af5c3
902c2c11-8ee0-41d8-8c78-1c9fe49af5c3
8ed352a3-d079-46a9-abf1-45d10e156397
8ed352a3-d079-46a9-abf1-45d10e156397
# и так далее
```

Теперь давайте остановим процесс-отправитель. В результате получатель будет простаивать и перестанет выводить данные:

```
pkill producer3.sh
[1]-  Terminated                  ./producer3.sh
```

Если повторно запустить `producer3.sh`, передача данных снова возобновится. С помощью этой процедуры мы продемонстрировали, что именованный канал остается устойчивым и надежным независимо от состояния процессов отправителя и получателя.

Именованные каналы — это, по сути, очереди, где данные обрабатываются и выбывают из очереди в том же порядке, в котором поступают в нее. Однако если через именованный канал взаимодействуют более двух процессов, подход FIFO может не подойти, особенно когда одни процессы требуют более высокого приоритета, чем другие. Позвольте представить вам сокеты — еще один механизм IPC.

Сокеты

Существует два механизма межпроцессного взаимодействия, основанных на сокетах:

- **Сокеты IPC**, которые также известны как **доменные сокеты Unix**.
- **Сетевые сокеты**, или **интернет-сокеты** на базе протоколов TCP и UDP.

Сокеты IPC используют локальный файл в качестве **адреса сокета**, благодаря чему процессы на одном узле могут обмениваться данными в обоих направлениях. А **сетевые сокеты** позволяют взаимодействовать процессам, которые выполняются не только на локальном компьютере, но и на разных узлах, — это осуществляется через службы TCP/UDP. Если отвлечься от очевидных различий в реализации, можно считать, что каналы передачи данных в случае сокетов IPC и сетевых сокетов ведут себя одинаково.

Оба типа сокетов функционируют как потоки, поддерживают двунаправленную связь и эмулируют модель «клиент — сервер». Канал связи сокета активен до тех пор, пока его не закроет одна из сторон, тем самым разорвав соединение IPC.

Давайте адаптируем нашу схему с отправителем и получателем так, чтобы передавать данные через доменный сокет Unix. Мы реализуем соответствующее соединение через **netcat** — мощный сетевой инструмент, с помощью которого можно читать и записывать данные через сокетные соединения с помощью TCP, UDP и IPC. Если в вашем дистрибутиве Linux **netcat** не установлен по умолчанию, его можно легко установить. Для этого в Ubuntu/Debian используйте такую команду:

```
sudo apt install netcat
```

В Fedora/RHEL используйте такую команду:

```
sudo dnf install nmap
```

Чтобы получить дополнительную информацию о **netcat**, обратитесь к соответствующей справочной странице (**man netcat**).

В следующем примере сценарий отправителя будет находиться в файле **producer4.sh**, а сценарий получателя — в файле **consumer4.sh**. Вот соответствующий код:

```
packt@neptune:~$ cat producer4.sh
#!/bin/bash
SOCKET="/var/tmp/ipc.sock"
rm -rf "${SOCKET}"
while true
do
    uuidgen;
    sleep 5s;
done | tee /dev/tty | nc -lU "${SOCKET}"
```

```
packt@neptune:~$ cat consumer4.sh
#!/bin/bash
SOCKET="/var/tmp/ipc.sock"
nc -U "${SOCKET}"
```

Отправитель выступает в роли сервера, иницилируя окончечную точку прослушивания `netcat` с помощью сокета IPC (последняя команда в файле `producer4.sh`):

```
nc -lU "${SOCKET}"
```

Здесь ключ `-l` задает режим прослушивания (чтобы получатель выступал в роли сервера), а ключ `-U` указывает на то, что мы работаем с доменным сокетом Unix. Получатель подключается к оконечной точке сервера `netcat` как клиент с помощью похожей команды (последняя строка в сценарии `consumer4.sh`). Стороны обмениваются данными через общий дескриптор файла сокета IPC (`/var/tmp/ipc.sock`).

Отправитель каждую секунду генерирует случайные строки UUID (цикл `while`). Его вывод перенаправляется в `stdout` с помощью `tee`, а затем передается в `netcat`.

Получатель принимает все сообщения (UUID), которые сгенерировал отправитель. Если запустить сценарий `consumer4.sh` сам по себе, он сообщит об ошибке: даже если файл сокета `ipc.sock` к этому моменту уже существует, он не подключен к другому концу и не может передавать данные:

```
packt@neptune:~$ ./consumer4.sh
nc: /var/tmp/ipc.sock: Connection refused1
```

Однако если сначала запустить сценарий отправителя, а затем — получателя, то данные начнут успешно передаваться. Если прервать процесс отправителя, то процесс получателя будет приостановлен.

В этом примере мы реализовали коммуникационную прослойку для сокета IPC с помощью `netcat`, но можно было использовать и другие инструменты — например, похожую утилиту `socat`.

В следующем разделе мы покажем вам несложные примеры того, как с помощью сценариев решать практические задачи администрирования Linux. Заодно вы узнаете, как создавать пакеты приложений из исходного кода.

¹ В соединении отказано. — *Примеч. пер.*

Как выполнять административные задачи с помощью сценариев

Сценарии в Linux чаще всего пишут для того, чтобы облегчить рутинные административные задачи. Сценарии позволяют системным администраторам автоматизировать рабочие процессы так, чтобы обслуживать систему было проще и комфортнее. Из всего многообразия сценариев командной оболочки мы приведем только несколько небольших примеров и надеемся, что они смогут продемонстрировать, как сценарии решают задачи системного администрирования.

По ходу дела мы покажем, как собрать программный пакет для конкретного дистрибутива из исходного кода. Мы упоминали об этом механизме в главе 4, но тогда вы еще не знали, как создать сценарий. Ранее в текущей главе мы использовали для примеров только Ubuntu 22.04.2 LTS, но с этого момента перейдем на серверную версию Fedora 37.

Как создавать сценарии для задач системного администрирования

В этом разделе мы создадим несколько сценариев для административных задач. Как уже говорилось в начале этой главы, сценарий оболочки — это последовательность команд Bash, которые хранятся в определенном файле и выполняются, если его запустить. В следующих подразделах мы рассмотрим три сценария для трех разных задач администрирования.

Сценарий для обновления системы

Давайте создадим сценарий, который обновляет систему в указанное время с помощью команды `dnf update`. Это очень незамысловатый сценарий, который выполняет простую команду и выводит несколько сообщений на стандартный вывод. Не забывайте, что на этот раз мы используем серверный дистрибутив Fedora. Проще всего будет запустить внутри сценария нужную команду:

```
sudo dnf update -y
```

Проблема в том, что для этого требуется пароль `sudo`. Это идет вразрез с идеей автоматизации, потому что пользователь должен указывать пароль вручную. Давайте узнаем, как решить эту проблему. Сначала посмотрим на код сценария:

```
[packt@saturn2 ~]$ cat /home/packt/update_script.sh
#!/bin/bash
# Простой сценарий для обновления системы
echo "Начинаем обновление системы..."
sudo dnf update -y
echo "Система обновлена. Приятного использования!"
```

Чтобы этот сценарий выполнялся без пароля `sudo`, нам понадобится отредактировать файл `/etc/sudoers`, как показано ниже.

Пожалуйста, имейте в виду, что этот метод затрагивает настройки безопасности на уровне целой системы, поэтому его следует использовать осторожно, понимая все возможные последствия.

Сначала раскомментируйте строку, которая начинается с `%wheel`:

```
%wheel  ALL=(ALL)          NOPASSWD: ALL
```

Затем замените `ALL` в конце этой строки на полный путь к нужной команде¹:

```
%wheel  ALL=(ALL)          NOPASSWD: /usr/bin/dnf update
```

В результате любой пользователь, который состоит в группе `wheel` (в нашем случае это текущий пользователь `packt`), сможет запускать сценарий с этой командой, не вводя пароль. Чтобы в этом убедиться, попробуйте запустить сценарий от имени обычного пользователя и посмотреть, появится ли запрос на пароль для `sudo`.

Теперь нужно запланировать запуск сценария на определенное время. Не забудьте сделать файл сценария исполняемым, чтобы его можно было запустить. Поскольку у нас серверный дистрибутив, мы предполагаем, что система работает круглосуточно без перерывов, поэтому запускать сценарий при загрузке системы не имеет смысла. Кроме того, мы хотим, чтобы система всегда оставалась в актуальном состоянии. Чтобы запланировать выполнение сценария, давайте воспользуемся `cron` и `crontab`. В следующем разделе мы покажем, как они работают.

Как планировать задачи в Linux с помощью cron

Планировщик заданий `cron` использует конфигурационные файлы в формате `crontab`, который поначалу может показаться пугающим. Но как только вы в нем разберетесь, вы начнете легко справляться с планированием в Linux. Возможно, самое сложное в `cron` — это писать определения задач, а особенно настраивать **расписание**. Однако в основном конфигурационном файле `/etc/crontab` приведена очень полезная схема того, как назначить задачу на определенное время:

```
# Пример определения задачи:
# .----- минуты (0-59)
# | .----- часы (0-23)
# | | .----- день месяца (1-31)
# | | | .----- номер месяца (1-12) ИЛИ jan, feb, mar, apr...
# | | | | .---- день недели (0-6) (вс=0 или 7)
# | | | | | ИЛИ sun, mon, tue, wed, thu, fri, sat
# * * * * * имя_пользователя команда, которую нужно выполнить
```

¹ Чтобы уточнить путь к `dnf` в своей системе, запустите `which dnf`. — *Примеч. ред.*

Мы надеемся, что эта схема не требует пояснений. Теперь давайте настроим новое задание `cron` для нашего сценария обновления. Для этого мы воспользуемся командой `crontab -e`, которая открывает конфигурационный файл текущего пользователя в текстовом редакторе оболочки по умолчанию — в нашем случае (в Fedora) это Vim. Команда `crontab -e` запускает новый экземпляр Vim, в котором нужно ввести такое задание:

```
00 23 * * 0 /home/packt/update_script.sh
```

Этот код приводит к тому, что наш сценарий обновления `update_script.sh` запускается каждое воскресенье в 23:00 от имени пользователя `packt`.

После того как вы сохранили файл, можно проверить, было ли создано соответствующее задание `cron`, — для этого служит команда `crontab -l`. Если вы хотите просмотреть задания определенного пользователя, можно использовать ключ `-u` — например, для пользователя `packt`:

```
sudo crontab -u packt -l
```

Попробуем улучшить определение задания `crontab` так, чтобы вывод нашего сценария и его сообщения об ошибках перенаправлялись в `/dev/null`. Это может быть полезно, если вы не хотите каждый раз видеть вывод команды или сообщения об ошибках в обычном терминале. Для этого строку нужно дополнить так:

```
00 23 * * * 0 /home/packt/update_script.sh > /dev/null 2>&1
```

`crontab` — это мощный инструмент, который наверняка пригодится вам для того, чтобы планировать задачи в Linux. Однако это не единственное средство такого характера; постарайтесь самостоятельно изучить другие доступные команды, например `at`.

Далее мы рассмотрим еще один короткий сценарий для резервного копирования файлов.

Сценарий для резервного копирования

Сценарии командной оболочки часто применяются, чтобы создавать **резервные копии** файлов. Давайте напишем такой сценарий, чтобы позже его можно было запланировать на то или иное время. Код сценария выглядит так:

```
[packt@saturn2 ~]$ cat backup_script.sh
```

```
#!/bin/bash
backup_dirs="/home/packt"    # Какие каталоги копировать
destination="/mnt/backup"    # Куда копировать

# Имя файла архива
date_backup=$(date +"%Y-%m-%d")
hostname=$(hostname)
file="$date_backup-$hostname.tgz"
```

```
printf "%s\n" "Создаем резервную копию $backup_dirs в $destination/$file"

# Выполняем резервное копирование с помощью tar
tar czf $destination/$file $backup_dirs

printf "%s\n" "Резервная копия готова"
ls -lh $destination          # Содержимое целевого каталога
```

Этот сценарий создает резервную копию каталога `/home/packt` в каталоге `/mnt/backup`. Содержимое исходного каталога архивируется с помощью команды `tar` и сохраняется в виде файла `.tgz`, имя которого состоит из текущей даты и имени узла.

Форматирование даты может сбить вас с толку, но очень полезно знать, как использовать эту возможность в сценариях. Чтобы получить дополнительную информацию, используйте команду `date --help`.

В следующем подразделе мы покажем, как создать простой генератор случайных паролей в Bash.

Сценарий для генератора случайных паролей

Безопасность ваших учетных записей в интернете или на локальном компьютере очень важна, поэтому стоит использовать надежные пароли. Рассмотрим пример сценария, который генерирует случайные пароли. Существует множество готовых генераторов паролей для Linux (например, `pwmake`), но мы подумали, что вам может быть интересно написать такой сценарий самостоятельно. Мы будем использовать утилиту `openssl` и стандарт кодирования `base64`. Узнать о нем подробнее можно на странице developer.mozilla.org/en-US/docs/Glossary/Base64. Не забывайте, что в Linux есть и много других способов создать случайные данные — например, генератор псевдослучайных чисел `/dev/urandom`. Изучите разные инструменты и выбирайте для каждой задачи тот, который лучше подходит. А так выглядит код сценария, который генерирует пароль:

```
#!/bin/bash

printf "\n"
printf "%s\n" "🔑 Генератор случайных паролей 🛡️"
printf "\n"
printf "%s" "Введите длину пароля (рекомендуется не менее 8 символов): "
read n
printf "%s" "Сколько паролей нужно сгенерировать? "
read p
printf "\n"

for (( i=0 ; i<$p ; i++ ))
do
    if [ $n -le 8 ]
```

```
then
    echo "Пароль слишком короткий. Попробуйте еще раз"
    exit 1
else
    openssl rand -base64 $n
fi
done

printf "\n"
```

Если запустить сценарий, он предложит указать нужную длину пароля и количество паролей, которые надо сгенерировать:

 Генератор случайных паролей 

```
Введите длину пароля (рекомендуется не менее 8 символов): 12
Сколько паролей нужно сгенерировать? 2
```

```
2ptU+026T7a6Ji+w
x/rmSyJwUX5DmVza
```

Создание сценариев — это чрезвычайно важная тема, которую вам следует освоить. Мы рассмотрели ее только на самом элементарном уровне, но надеемся, что материал этой главы послужил надежным фундаментом, на основе которого вы начнете писать сценарии для системного администрирования.

В следующем подразделе мы расскажем, как упаковывать сценарии в полноценные приложения, которые можно установить из командной строки Linux. Это дополнит главу 3, из которой вы узнали, как управлять пакетами Linux.

Как создавать пакеты для сценариев

Сценарии командной оболочки — это полноценное программное обеспечение, поэтому их, как и любые другие совместимые программы, можно развернуть в системе Linux в виде пакетов для той или иной платформы. В этом разделе мы дополним материал главы 3 и продемонстрируем, как создать пакет из сценария Bash. Аналогичным образом вы сможете упаковывать программы, которые написаны на других языках (в Linux это чаще всего C/C++, Python, Rust, Java или Go). В следующем подразделе мы разберемся, как создать пакет RPM для дистрибутивов на базе RHEL. В практических примерах мы будем использовать Fedora Linux 37.

Как создавать пакеты RPM из исходного кода

Чтобы создать пакет RPM, мы возьмем сценарий генератора паролей, который написали в предыдущем разделе. Прежде чем приступить к практической работе, давайте вкратце вспомним о том, какие бывают языки программирования и какое место среди них занимает Bash.

Чаще всего разработчики пишут программы в виде человекочитаемого исходного кода на том или ином языке программирования. Чтобы компьютер понял этот код, его нужно перевести в машинный код. Большинство языков программирования относятся к одной из двух больших категорий: **интерпретируемые языки** (такие как Python или язык командной оболочки Bash) и **компилируемые языки** (такие как C/C++, Java и Go). Код сценария Bash выполняется строка за строкой, без предварительного преобразования сценария в машинный код. Подробнее о классификациях языков программирования можно узнать на сайте en.wikipedia.org/wiki/List_of_programming_languages_by_type.

Часто исходный код программного обеспечения распространяется в виде сжатых архивов (файлы `.tar.gz`), которые затем помещаются в пакеты RPM. Архив обычно состоит из файла исходного кода и файла лицензии. В случае **свободного и открытого программного обеспечения** чаще всего применяются лицензии GPLv3 или LGPLv3, но есть и много других — например, лицензия MIT или лицензии Apache и BSD.

Подробную информацию о типах лицензий для свободного и открытого ПО можно найти на странице en.wikipedia.org/wiki/Comparison_of_free_and_open-source_software_licenses.

Давайте перейдем от теории к практике и превратим наш сценарий Bash в полноценный пакет RPM:

1. **Создайте каталог.** Сначала нам понадобится каталог, в котором будут храниться все нужные файлы. Создайте в домашнем каталоге новый каталог `passgen-0.1`.
2. **Создайте файл лицензии¹.** Для нашего сценария Bash мы создадим файл лицензии, который будет входить в пакет. Назовите файл `LICENSE.txt` и поместите в каталог, который создали на предыдущем шаге. Мы будем использовать лицензию GPLv3 с открытым исходным кодом. О том, какой именно текст нужно поместить в файл лицензии, можно узнать на странице gnu.org/licenses/gpl-howto.html.
3. **Поместите файлы в новый каталог.** Поместите файл `LICENSE.txt` в каталог `passgen-0.1`, а также скопируйте в этот каталог сценарий `passgen_script.sh` и переименуйте его в `passgen-0.1.sh`.
4. **Создайте архив с исходными файлами.** Поместите каталог `passgen-0.1` в архив `tar.gz` с помощью такой команды:

```
tar -cvzf passgen-0.1.tar.gz passgen-0.1
```

¹ В описанном процессе файл лицензии технически не обязателен. — *Примеч. ред.*

(В этом примере мы запускаем все команды из домашнего каталога. Если вы находитесь в другом каталоге, то в некоторых командах понадобится изменить некоторые относительные пути.)

5. **Установите утилиты для работы с RPM.** Чтобы создавать пакеты RPM, нам понадобятся специальные утилиты, например `rpmdevtools`. Если она еще не установлена, установите ее с помощью такой команды:

```
sudo dnf install rpmdevtools
```

В результате у вас появятся все пакеты и зависимости, которые позволяют настраивать рабочую область для создания пакетов.

6. **Настройте рабочую область.** Чтобы создать рабочую область RPM, запустите такую команду:

```
rpmdev-setuptree
```

В результате в вашем домашнем каталоге появится новый каталог `rpmbuild` с такими подкаталогами:

```
[packt@saturn2 ~]$ tree rpmbuild/
rpmbuild
├── BUILD      # артефакты сборки
├── RPMS       # двоичные пакеты RPM
├── SOURCES    # исходные файлы
├── SPECS      # спецификации сборки (файлы .spec)
└── SRPMS      # пакеты RPM с исходным кодом
```

7. **Скопируйте файл `.tar.gz` в каталог `SOURCES`.** Добавьте архив, который вы создали на шаге 4, в каталог `~/rpmbuild/SOURCES`:

```
cp passgen-0.1.tar.gz rpmbuild/SOURCES/
```

Теперь исходные файлы для нашего пакета находятся в нужном месте.

8. **Создайте спецификацию сборки.** Спецификация сборки — это файл (обычно с расширением `.spec`), который описывает, как нужно собирать и упаковывать приложение. Запустите команду `rpmdev-newspec`, чтобы создать простой шаблон файла спецификации в каталоге `SPECS`:

```
rpmdev-newspec passgen
```

Просмотрите содержимое каталога `~/rpmbuild/SPECS/`, чтобы убедиться, что в нем появился файл `passgen.spec`. Этот файл можно создать и вручную, без `rpmdev-newspec`.

В этой книге мы не будем подробно объяснять синтаксис спецификации. Ее ключевые элементы — макросы, которые начинаются со знака `%` (например, `%files` или `%(name)`). Формат файлов `.spec` подробно описан в официальной документации по RPM: rpm-software-management.github.io/rpm/manual/spec.html.

Файл `passgen.spec` содержит строки, которые автоматически сгенерировала утилита `rpmdev-newspec`, но нам понадобится отредактировать его под наши нужды. В частности, мы удалим все директивы, которые относятся к компиляции, потому что наш сценарий `passgen-0.1.sh` не нужно компилировать. Окончательная спецификация сборки выглядит так:

```
Name:          passgen
Version:       0.1
Release:       1%{?dist}
Summary:       Bash password generator1
BuildArch:     noarch

License:       GPLv3+
Source0:       %{name}-%{version}.tar.gz

Requires:      bash

%description
A simple password generator written as Bash shell script

%prep
%setup -q

%install
rm -rf $RPM_BUILD_ROOT
mkdir -p $RPM_BUILD_ROOT/%{_bindir}
cp %{name}-%{version}.sh $RPM_BUILD_ROOT/%{_bindir}

%files
%{_bindir}/%{name}-%{version}.sh

%changelog
* Sun Apr 16 2023 packt
- Test release
```

9. **Соберите пакет RPM.** Команда `rpmbuild` позволяет создать готовый пакет RPM. В этом примере мы создадим двоичный пакет на основе спецификаций сборки из файла `passgen.spec`, но возможны и другие варианты. Например, можно создать пакет RPM с исходным кодом (**SRPM**), из которого пользователь может заново скомпилировать и развернуть приложение на своей машине.

Чтобы создать двоичный пакет RPM, запустите `rpmbuild` с ключом `-bb`:

```
rpmbuild -bb rpmbuild/SPECS/passgen.spec
```

¹ Описание программ и другие сведения в файле спецификации традиционно принято писать по-английски. — *Примеч. пер.*

Чтобы убедиться, что сборка прошла успешно, просмотрите содержимое каталога RPMS. В нем должен появиться новый каталог `noarch`¹, в котором содержится готовый пакет RPM с нашим генератором паролей:

```
[packt@saturn2 ~]$ tree ~/rpmbuild/RPMS/  
/home/packt/rpmbuild/RPMS/  
└─ noarch  
    └─ passgen-0.1-1.fc37.noarch.rpm
```

Вот вы и собрали свой первый двоичный пакет RPM из файла сценария Bash! Похожим образом можно создавать пакеты из исходного кода на любом языке, с которым вы работаете. Сборка пакетов — не очень сложная задача, а если у вас под рукой будет хорошая документация по макросам, которые используются в файле спецификации, у вас все получится. Некоторые ресурсы на эту тему перечислены в разделе «Дополнительная литература» в конце этой главы.

Итоги

Из этой главы вы узнали, как создавать сценарии командной оболочки Bash в Linux. Вы знаете, что такое переменные оболочки и переменные окружения, а также умеете использовать в сценариях такие структуры программирования, как циклы, условия и массивы. Наконец, вы на практических примерах наблюдали, как работает межпроцессное взаимодействие. Полученные навыки помогут вам создавать собственные сценарии в любом дистрибутиве Linux. При этом вам пригодится также опыт работы с текстовыми редакторами. Все, что вы узнали о сценариях Bash, станет неоценимым подспорьем во многих аспектах вашей повседневной работы в качестве системного администратора.

В следующей главе вы узнаете, как управлять безопасностью в Linux и механизмами контроля доступа, а также познакомитесь с AppArmor, SELinux и межсетевыми экранами. Соответствующие знания и умения важны для каждого администратора и входят в любую программу профессионального обучения в области Linux.

¹ Сокр. по architecture (без определенной архитектуры). Обычно в каталоге RPMS размещаются отдельные подкаталоги со сборками для той или иной целевой архитектуры — например, `x86_64` или `arm64`. В подкаталоге `noarch` находятся пакеты, которые не зависят от архитектуры и должны быть совместимы с любой системой. Кроме сценариев командной оболочки, к таким ресурсам относятся, например, текстовые файлы или медиафайлы, которые не предполагают отдельных версий для разных архитектур. — *Примеч. ред.*

Вопросы

В этой главе мы рассмотрели сценарии командной оболочки Bash в системах семейства Linux. Вот несколько вопросов, которые помогут вам проверить свои знания и попрактиковаться:

1. Что такое массивы и как использовать их в сценариях Bash? Данные какого характера могут храниться в массивах?

Подсказка: вспомните, какие разновидности массивов бывают в Bash и как они определяются.

2. Что такое бесконечный цикл?

Подсказка: вспомните примеры из этой главы, где использовались бесконечные циклы. Как выйти из бесконечного цикла, если вы оказались в нем по ошибке?

3. Какие типы циклов можно использовать в сценариях Bash?

Подсказка: этих типов не так много, но они не ограничиваются циклами `for`.

4. Как собрать двоичный пакет в формате DEB?

Подсказка: ознакомьтесь со справочными страницами `debuild` и других утилит из пакета `devscripts`, а также прочтите руководство для разработчиков ПО для Debian по адресу debian.org/doc/manuals/maint-guide.

Дополнительная литература

Чтобы получить дополнительную информацию обо всем, что рассматривалось в этой главе, обратитесь к следующим ресурсам:

- Ведран Дакич (Vedran Dakic), Жасмин Реджепагич (Jasmin Redzepagic). «Linux Command Line and Shell Scripting Techniques».
- Документация на официальных сайтах:

Red Hat: docs.redhat.com/en/documentation/red_hat_enterprise_linux/9/html/packaging_and_distributing_software/

Fedora: docs.fedoraproject.org/en-US/packaging-guidelines/rpm.org

9

Как обеспечивать безопасность в Linux

Чтобы обеспечивать **безопасность** в системе Linux, обычно приходится искать баланс между разными требованиями и ограничениями. В любом случае конечная цель состоит в том, чтобы защищать данные от нежелательного доступа. Хотя достичь этого можно по-разному, мы стремимся использовать такие методы, которые обеспечивают максимальную защиту, а также позволяют наиболее эффективно администрировать систему. Как правило, стоит начать с того, чтобы оценить потенциальную поверхность атак и уязвимостей — как внутреннего, так и внешнего характера. После этого, как в рыцарских романах, пора возводить стены (не слишком высокие) и надевать доспехи (не слишком тяжелые). В нашем случае внешние стены — это **межсетевой экран (брандмауэр)**, а внутри, на уровне системы, мы внедряем **политики безопасности приложений**. В этой главе мы познакомимся с обоими механизмами, но как найти баланс — в конечном итоге решать только вам.

В первой части этой главы мы рассмотрим **системы управления доступом (ACM)** и соответствующие модули безопасности — **SELinux** и **AppArmor**, а во второй части поговорим о том, как фильтровать пакеты и настраивать межсетевые экраны.

К концу этой главы вы научитесь налаживать и поддерживать инструменты безопасности приложений и межсетевые экраны. Это станет первым уверенным шагом к тому, чтобы обеспечивать безопасность в системах Linux.

В этой главе рассматриваются такие темы:

- Основы безопасности Linux: какие системы управления доступом присутствуют в ядре Linux.
- Как устроена система управления доступом SELinux и как управлять политиками контроля доступа с ее помощью.
- Модуль безопасности AppArmor: как контролировать активность приложений на основе профилей безопасности.
- Межсетевые экраны: всесторонний обзор соответствующих модулей и утилит, включая **Netfilter**, **iptables**, **nftables**, **firewalld** и **ufw**.

Технические требования

Эта глава охватывает довольно обширный круг вопросов, и, чтобы полноценно их рассмотреть, понадобится много практиковаться в командной строке. Для этих упражнений мы рекомендуем использовать Fedora или Ubuntu с доступом через терминал или по SSH. Лучше, чтобы у вас был прямой консольный доступ — на тот случай, если вы вдруг неверно настроите межсетевой экран.

Как устроена безопасность в Linux

Чтобы обеспечивать безопасность компьютерной системы или сети, необходим целый комплекс мер и инструментов. Среди них — средства, с помощью которых системные администраторы контролируют доступ пользователей и процессов к различным ресурсам — например, файлам, устройствам и интерфейсам — в одной или разных системах. Ядро Linux предоставляет несколько таких механизмов, которые в совокупности называются системами управления доступом (АСМ). Давайте кратко перечислим их:

- **Избирательное управление доступом (DAC)** распространяется на объекты файловой системы, такие как файлы, каталоги и устройства. Владелец объекта регулирует доступ к нему с помощью прав доступа (разрешений). При избирательном управлении доступ к *объектам* предоставляется на основе идентификации пользователей и групп (*субъектов*). В зависимости от полномочий субъекта он может передавать права другим субъектам — например, так происходит, когда администратор управляет правами доступа обычных пользователей.
- **Списки управления доступом (ACL)** позволяют контролировать, у каких субъектов (например, пользователей и групп) есть доступ к тем или иным объектам файловой системы (например, файлам и каталогам).
- **Мандатное (обязательное) управление доступом (MAC)** ограничивает возможность субъектов управлять доступом к объектам, которыми они владеют. В отличие от чистого DAC, где объекты файловой системы находятся в полном распоряжении пользователей-владельцев, MAC добавляет ко всем объектам дополнительные категории — так называемые метки доступа, которые ограничивают возможность субъектов взаимодействовать с теми или иными объектами (в том числе собственными). Стандартным механизмом MAC в RHEL и Fedora служит SELinux, а в Ubuntu, Debian и openSUSE — AppArmor.
- **Ролевое разграничение доступа (RBAC)** — это, по сути, расширенная форма MAC. Пользователям системы назначаются определенные *роли*, от которых зависит доступ к тем или иным объектам файловой системы. Роли могут опираться на бизнес-логику или функциональные критерии, и разным ролям могут соответствовать разные уровни доступа к объектам.

В отличие от DAC или MAC, где доступ субъектов к объектам определяется исключительно соответствующими правами доступа, модель RBAC служит

логической абстракцией над MAC или DAC, потому что субъекты должны состоять в определенной группе или исполнять определенную роль, чтобы взаимодействовать с объектами.

- **Многоуровневая безопасность (MLS)** — это еще одна особая форма MAC, в которой *субъектами* являются процессы, а *объектами* — файлы, сокеты и другие системные ресурсы.
- **Мультикатегорийная безопасность (MCS)** — это метод управления доступом в SELinux, который позволяет пользователям пометить файлы *категориями*. MCS в значительной мере задействует возможности MLS в SELinux.

Завершая обзор механизмов ACM, отметим, что мы уже рассмотрели некоторые средства DAC и ACL в главе 4 «Как управлять пользователями и группами», в частности в разделе «Как управлять правами доступа».

Далее мы обратимся к модулю SELinux, который реализует MAC и другие политики контроля доступа.

Система управления доступом SELinux

SELinux (Security-Enhanced Linux¹) — это система безопасности, которая работает на уровне ядра Linux и позволяет управлять политиками контроля доступа к системным ресурсам. Она поддерживает комбинацию моделей MAC, RBAC и MLS, о которых шла речь в предыдущем разделе. В SELinux входят модули безопасности в пространстве ядра и утилиты командной строки в пространстве пользователя, с помощью которых системные администраторы могут контролировать, *кто и к чему* получает доступ в системе. SELinux также позволяет защитить систему от небезопасных конфигураций и потенциально скомпрометированных процессов.

В основе SELinux лежат принципы, которые разработало **Агентство национальной безопасности США**. В 2000 году SELinux был выпущен для сообщества разработчиков как набор модулей безопасности для ядра Linux (LSM) с открытым исходным кодом, а в 2003 году он был включен в ядро Linux (начиная с версии 2.6).

В следующем разделе мы рассмотрим, как устроен SELinux. Для всех примеров мы будем использовать серверную версию Fedora 37.

Как устроен SELinux

В SELinux используются **политики безопасности**, которые определяют различные уровни управления доступом для приложений, процессов и файлов в системе. Политика безопасности — это набор правил, которые описывают, к чему можно или нельзя получить доступ.

¹ Linux с расширенной безопасностью. — *Примеч. пер.*

SELinux работает с **субъектами** и **объектами**. Когда то или иное приложение или процесс (*субъект*) запрашивает доступ к файлу (*объекту*), SELinux проверяет необходимые права доступа, которые относятся к запросу, и применяет соответствующие механизмы контроля. Права доступа для субъектов и объектов хранятся в таблице **AVC**¹, которая формируется на основе базы данных политик SELinux.

Типичная политика безопасности SELinux состоит из нескольких ресурсов (файлов), каждый из которых отражает определенный аспект политики:

- **Принудительное присвоение типов.** Действия, которые разрешает или запрещает политика (например, доступ к файлу на чтение или запись).
- **Интерфейс.** Прикладной интерфейс, с которым взаимодействует политика (например, протоколирование).
- **Файловые контексты.** Системные ресурсы, связанные с политикой (например, файлы журналов).

Чтобы создать конкретную политику безопасности, эти файлы компилируются с помощью инструментов сборки SELinux. Готовая политика загружается в ядро, добавляется в базу данных политик SELinux и вступает в действие без перезагрузки системы.

Создавая политики SELinux, обычно сначала тестируют их в *уведомительном* режиме, когда нарушения регистрируются, но все равно допускаются. Если выявлены нарушения, на помощь приходит утилита **audit2allow** из набора инструментов SELinux. Она создает трассировки журнала, на основе которых можно добавить новые правила, чтобы политика не запрещала добросовестный доступ. Нарушения политик SELinux записываются в журнал `/var/log/messages` и снабжаются префиксом `avc: denied`.

Прежде чем узнать, как настраивать политики безопасности SELinux и управлять ими, давайте рассмотрим некоторые операции более высокого уровня, которые позволяют взаимодействовать с SELinux в повседневных задачах администрирования.

Режимы SELinux

В каждый момент времени SELinux в системе либо *включен*, либо *отключен*. Когда он включен, он работает в одном из следующих режимов:

Enforcing	Принудительный	SELinux обеспечивает, чтобы политики безопасности соблюдались. В RHEL/Fedora этот режим включен по умолчанию
Permissive	Уведомительный	Система отслеживает нарушения политик безопасности, но не блокирует несанкционированный доступ. Нарушения заносятся в <code>/var/log/messages</code>

¹ Сокр. Access Vector Cache (кэш векторов доступа). — *Примеч. пер.*

Когда SELinux отключен, политики безопасности не отслеживаются и не применяются. Команда `sestatus` позволяет узнать текущее состояние SELinux:

```
[packt@saturn2 ~]$ sestatus
SELinux status:                enabled
SELinuxfs mount:              /sys/fs/selinux
SELinux root directory:       /etc/selinux
Loaded policy name:            targeted
Current mode:                  enforcing
Mode from config file:         enforcing
Policy MLS status:             enabled
Policy deny_unknown status:    allowed
Memory protection checking:    actual (secure)
Max kernel policy version:     33
```

Когда SELinux включен, команда `getenforce` без аргументов позволяет узнать текущий режим (Enforcing или Permissive).

Чтобы переключиться в уведомительный режим, запустите такую команду:

```
sudo setenforce 0
```

После этого `getenforce` выведет `Permissive`. С помощью похожей команды можно переключиться обратно в принудительный режим:

```
sudo setenforce 1
```

Чтобы регулировать режим SELinux, можно также присвоить соответствующее значение (`enforcing` или `permissive`) параметру `SELINUX` в конфигурационном файле `/etc/selinux/config`.

Если вы вручную редактируете конфигурационный файл SELinux, то нужно перезагрузить систему, чтобы изменения вступили в силу.

Когда SELinux включен, с помощью параметра `SELINUXTYPE` в конфигурационном файле можно задать один из следующих уровней политики SELinux: `targeted`, `minimum` или `mls`. Эти значения подробно описаны в комментариях в том же файле.

По умолчанию в SELinux установлен уровень политики `targeted`, и обычно не рекомендуется переключать эту настройку, если только не применяется MLS.

Уровень `targeted` означает, что политики распространяются на *целевые* процессы, которые специально указал администратор. Как правило, это процессы, которые рассматриваются как наиболее вероятная цель атаки, например системные демоны (такие, как `dhcpcd` и `sshd`) и общеизвестные серверные приложения (такие, как `Apache` и `PostgreSQL`). Целевым процессам присваивается определенный тип, благодаря чему они помещаются в *ограниченный* домен. SELinux

отслеживает активность процессов из этого домена и ограничивает их доступ к ресурсам, которые может использовать злоумышленник. Все остальные (нецелевые) процессы работают без специальных ограничений и обычно помечаются типом `unconfined_t`.

Чтобы полностью отключить SELinux, можно открыть файл `/etc/selinux/config` в вашем любимом текстовом редакторе и внести такое изменение¹:

```
SELINUX=disabled
```

Внести изменения в конфигурационный файл можно также из командной строки. Например, такая команда изменит режим SELinux с `Enforcing` на `Disabled`:

```
sudo sed -i 's/SELINUX=enforcing/SELINUX=disabled/g' /etc/selinux/config
```

Чтобы просмотреть текущую конфигурацию, можно вывести содержимое конфигурационного файла с помощью `cat`:

```
cat /etc/selinux/config
```

Чтобы изменения вступили в силу, необходимо перезагрузить систему:

```
sudo systemctl reboot (или просто sudo reboot)
```

Теперь давайте разберемся, как система принимает решения по управлению доступом. Для этого мы поговорим о **контекстах SELinux**.

Контексты SELinux

Когда SELinux включен, процессы и файлы помечаются **контекстом**, который содержит дополнительную информацию, например сведения о *пользователе*, *роль*, *тип* и *уровень* (необязательно). На основе этой информации SELinux принимает решения по управлению доступом.

SELinux добавляет к командам `ls`, `ps` и другим ключ `-Z`, который позволяет отображать контекст безопасности для объектов файловой системы, процессов и других ресурсов.

Давайте создадим произвольный файл и изучим соответствующий контекст SELinux:

```
[packt@saturn2 ~]$ touch afile
[packt@saturn2 ~]$ ls -Z afile
unconfined_u:object_r:user_home_t:s0 afile
```

¹ В современных системах эта настройка обычно не отключает SELinux полностью, а только деактивирует все политики; этого достаточно для большинства реальных задач администрирования. Чтобы эффективно отключить SELinux, можно задать параметр командной строки ядра `selinux=0` (например, с помощью `grubby`). — *Примеч. ред.*

Контекст SELinux состоит из четырех полей, разделенных двоеточиями (:):

пользователь : роль : тип : уровень

Давайте рассмотрим эти поля подробнее:

- **Пользователь SELinux.** Это идентификатор, известный политике, который авторизован для определенного набора ролей и которому соответствует определенный уровень, заданный диапазоном MLS/MCS (подробнее см. «Уровень SELinux» далее в этом списке). Политика SELinux связывает с каждой учетной записью Linux соответствующий идентификатор пользователя SELinux. Это позволяет обычным пользователям Linux наследовать ограничения политик, которые относятся к пользователям SELinux.

Процесс, которым владеет пользователь Linux, получает идентификатор соответствующего пользователя SELinux, что позволяет задействовать его роли и уровни SELinux.

Команда `semanage login -l` отображает соответствие между учетными записями Linux и пользователями SELinux и должна запускаться с привилегиями суперпользователя. Если в вашей системе нет этой команды, установите ее в составе пакета `polycoreutils`. Вот что она выводит в нашем случае:

```
[packt@saturn2 ~]$ sudo semanage login -l
```

Login Name	SELinux User	MLS/MCS Range	Service
__default__	unconfined_u	s0-s0:c0.c1023	*
root	unconfined_u	s0-s0:c0.c1023	*

Чтобы получить дополнительную информацию о `semanage`, обратитесь к соответствующей справочной странице (`man semanage` или `man semanage-login`).

- **Роль SELinux.** Роли SELinux — это, по сути, атрибуты модели безопасности RBAC. В иерархии контекстов SELinux пользователи авторизуются для ролей, а роли — для типов или доменов. В этой терминологии **типы** характеризуют объекты файловой системы, а **домены** относятся к процессам.

Рассмотрим, например, процессы Linux. Роль SELinux выступает как промежуточный уровень доступа между доменами и пользователями SELinux. *Доступная* роль определяет, к какому домену (то есть к каким процессам) через нее можно получить доступ. В конечном счете этот механизм управляет тем, к каким типам объектов можно получить доступ, и тем самым минимизирует поверхность атак, которые связаны с повышением привилегий.

- **Тип SELinux** — это атрибут такой конструкции безопасности MAC, как *принудительное присвоение типов*. Политики безопасности SELinux контролируют, как те или иные типы могут обращаться друг к другу — либо когда домен получает доступ к типу, либо когда один домен получает доступ к другому.

- **Уровень SELinux** — это атрибут схемы MLS/MCS и необязательное поле в контексте SELinux. Это поле представляется в таком формате:

степени_конфиденциальности : категории

- *Степень конфиденциальности* можно понимать как степень секретности — например, «для общего пользования», «секретно» или «совершенно секретно». По умолчанию политика MLS в SELinux предусматривает 16 степеней конфиденциальности, которые обозначаются от **s0** (низшая степень) до **s15** (высшая степень). *Категория* — это необязательное дополнение, которое обозначается от **c0** до **c1023** и позволяет классифицировать ресурсы: например, **c1** может соответствовать финансовым документам, а **s12** — резервным копиям.
- В контексте SELinux степени конфиденциальности и категории выражаются в виде диапазонов, так что целый уровень может выглядеть, например, так: **s1-s3:c10.c15**. Если степень конфиденциальности всего одна, она повторяется, например: **s1-s1:c10**.

Теперь вы знаете, что такое контексты SELinux, и пора увидеть их в действии, начиная с контекстов для пользователей.

Контексты SELinux для пользователей

Команда **id -Z** выводит контекст SELinux, который связан с текущим пользователем, например:

```
[packt@saturn2 ~] $ id -Z
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
```

Реализация SELinux в RHEL/Fedora по умолчанию не накладывает на пользователей специальных ограничений, и поля контекста выглядят так:

unconfined_u	Пользователь
unconfined_r	Роль
unconfined_t	Тип
s0-s0	Степень конфиденциальности
c0.c1023	Все категории (от c0 до c1023)

Далее мы рассмотрим контексты SELinux для процессов.

Контексты SELinux для процессов

Команда **ps -eZ** выводит контексты SELinux для текущих процессов. Например, так можно вывести контексты для процессов, связанных с SSH:

```
[packt@saturn2 ~]$ ps -eZ | grep sshd
system_u:system_r:sshd_t:s0-s0:c0.c1023      794 ? 00:00:00 sshd
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 924 ? 00:00:00 sshd
```

Здесь первая строка относится к серверному процессу **sshd**, который запущен с идентификатором пользователя **system_u**, ролью **system_r** (это стандартные

значения для системных демонов) и доменом `sshd_t`. А вторая строка отражает сеанс SSH текущего пользователя с неограниченным контекстом. Прежде чем завершить этот раздел о контекстах SELinux, давайте рассмотрим весьма пространственный сценарий доменного перехода, когда процесс из одного домена получает доступ к объекту (или процессу) из другого домена.

Доменные переходы в SELinux

Когда процесс в одном домене запрашивает доступ к объекту (или процессу) в другом домене, который защищен SELinux, мы имеем дело с **доменными переходами**. Если нет специальной политики безопасности, которая разрешает соответствующий переход, SELinux откажет в доступе.

Давайте проиллюстрируем доменный переход на простом примере, когда пользователь меняет свой пароль с помощью утилиты `passwd`. Для этого процесс `passwd` обращается к файлу `/etc/shadow` (а также, возможно, к `/etc/gshadow`). После того как пользователь вводит и подтверждает пароль, `passwd` хеширует его и сохраняет хеш-сумму в `/etc/shadow`.

Рассмотрим контексты упомянутых файлов:

```
[packt@saturn2 ~]$ ls -Z /usr/bin/passwd
system_u:object_r:passwd_exec_t:s0 /usr/bin/passwd
```

```
[packt@saturn2 ~]$ ls -Z /etc/shadow
system_u:object_r:shadow_t:s0 /etc/shadow
```

Утилита `passwd` помечена типом `passwd_exec_t`, а файл `/etc/shadow` — типом `shadow_t`. Должна существовать определенная цепочка политик безопасности, которая позволяет переходить от `passwd_exec_t` к `shadow_t`; иначе `passwd` не будет работать так, как ожидается.

Давайте убедимся, что это так. Чтобы просматривать политики безопасности, мы воспользуемся утилитой `sesearch`, которая выполняет поиск по базе данных политик SELinux. Она не установлена по умолчанию в Fedora, но ее можно установить в составе пакета `setools-console`:

```
sudo dnf install setools-console -y
```

После этого запустите такую команду, чтобы найти нужную политику:

```
sudo sesearch -s passwd_t -t shadow_t -p write --allow
```

Каждый ключ этой команды выбирает правила политики, которые удовлетворяют определенному условию:

<code>-s passwd_t</code>	Правила, в которых тип <code>passwd_t</code> указан в качестве источника
<code>-t shadow_t</code>	Правила, в которых тип <code>shadow_t</code> указан в качестве целевого типа
<code>-p write</code>	Правила, которые регулируют право на <i>запись</i>
<code>--allow</code>	Правила, которые <i>разрешают</i> операции, указанные с помощью <code>-p</code> (в данном случае операции записи)

Запустив команду, можно убедиться, что этому фильтру соответствует ровно одно правило, и среди его разрешений есть `append` и `write`, которые позволяют записывать в файл типа `shadow_t`:

```
[packt@saturn2 ~]$ sudo sestatus -s passwd_t -t shadow_t -p write --allow
allow passwd_t shadow_t:file { append create getattr ioctl link lock map open
read relabelfrom relabelto rename setattr unlink watch watch_reads write };
```

Обратите внимание, что мы немного слукавили и указали в качестве источника тип `passwd_t`, а не `passwd_exec_t`, к которому относится утилита `passwd`. Нам понадобится найти еще одно недостающее звено — политику, которая связывает `passwd_t` и `passwd_exec_t`. Это можно сделать так:

```
[packt@saturn2 ~]$ sudo sestatus -s passwd_t -t passwd_exec_t -p entrypoint
--allow
allow passwd_t passwd_exec_t:file { entrypoint execute getattr ioctl lock map
open read };
```

Это правило означает, что файлы типа `passwd_exec_t` (к которым относится наш `/usr/bin/passwd`) служат точкой входа (`entrypoint`) в домен `passwd_t`. А предыдущее правило разрешает процессам из этого домена обращаться к файлам из домена `shadow_t` и, в частности, модифицировать их. Теперь все сошлось.

Если вы не уверены, по каким полям искать нужные политики SELinux, можно запросить сразу все политики и отфильтровать вывод с помощью `grep`. Например, так можно вывести все разрешительные политики, в которых встречается одновременно `shadow_t` и `passwd_t`:

```
[packt@saturn2 ~]$ sestatus -A | grep shadow_t | grep passwd_t
allow passwd_t shadow_t:file { append create getattr ioctl link lock map open
read relabelfrom relabelto rename setattr unlink watch watch_reads write };
```

```
allow sysadm_passwd_t shadow_t:file { append create getattr ioctl link lock open
read relabelfrom relabelto rename setattr unlink watch watch_reads write };
```

Здесь ключ `-A` означает, что нужно искать именно разрешительные политики (в том числе типа `allow`). Подробнее о ключах и других особенностях команды `sestatus` можно прочитать на ее справочной странице (`man sestatus`).

Утилита `sestatus` очень удобна, когда нужно запрашивать политики безопасности, но системному администратору часто требуется также устранять неполадки или управлять конфигурацией и политиками SELinux. Для этого есть отдельные инструменты — например, `semanage`, который мы рассмотрим далее в разделе «Как управлять политиками SELinux». Но сначала давайте поговорим о том, как создать политику безопасности SELinux.

Как создать политику безопасности SELinux

Для примеров в этом разделе нам придется разработать и скомпилировать небольшую программу на языке C, чем мы пока не занимались в этой книге. Что-

бы компилировать код на C, в системе нужен набор компиляторов **GCC**, который можно установить в Fedora с помощью такой команды:

```
sudo dnf install gcc
```

Давайте создадим демон **packtd** и настроим политику SELinux так, чтобы он мог обращаться к **/var/log/messages**. Наш демон будет устроен очень просто: он всего лишь периодически открывает файл **/var/log/messages** для записи. Создайте в своем любимом текстовом редакторе (например, **nano**) файл **packtd.c** и добавьте в него такой код на C:

```
#include <unistd.h>
#include <stdio.h>

FILE *f;
char LOG_FILE[] = "/var/log/messages";

int main(void) {
    while (1) {
        f = fopen(LOG_FILE, "w");
        sleep(10);
        fclose(f);
    }
}
```

Запустите **gcc** с ключом **-o**, чтобы скомпилировать исходный код и получить двоичный исполняемый файл **packtd**:

```
[packt@saturn2 ~]$ gcc -o packtd packtd.c

[packt@saturn2 ~]$ ls -l | grep packt
-rwxr-xr-x 1 packt packt      16160 Mar  8 19:04 packtd
-rw-r--r-- 1 packt packt       202 Mar  8 19:29 packtd.c
```

После того как мы скомпилировали код, можно создать демон **packtd** и настроить соответствующую политику безопасности SELinux. Это упражнение не только продемонстрирует, как администрировать SELinux, но и научит вас настраивать системные демоны. Если вам нужно освежить в памяти информацию о демонах, пожалуйста, обратитесь к главе 5. Теперь давайте поговорим о том, как установить демон и настроить политику.

Как установить и настроить демон

Прежде всего нужно создать файл модуля подсистемы инициализации **systemd** для демона **packtd**. Откройте свой любимый текстовый редактор (например, **nano**) и создайте в нем файл под именем **packtd.service**:

```
[packt@saturn2 ~]$ cat packtd.service
[Unit]
Description="Проверка журнала"

[Service]
Type=simple
```

```
ExecStart=/usr/local/bin/packtd
```

```
[Install]
```

```
WantedBy=multi-user.target
```

Скопируйте исполняемый файл и файл модуля в подходящие места, например¹:

```
sudo cp packtd /usr/local/bin/
sudo cp packtd.service /usr/lib/systemd/system/
```

Теперь можно запустить демон `packtd` и убедиться, что он работает:

```
[packt@saturn2 ~]$ sudo systemctl start packtd
[packt@saturn2 ~]$ sudo systemctl status packtd
● packtd.service - "Проверка журнала"
   Loaded: loaded (/usr/lib/systemd/system/packtd.service; disabled; preset:
d>
   Active: active (running) since Wed 2023-04-19 21:34:25 EST; 5s ago
   Main PID: 1365 (packtd)
     Tasks: 1 (limit: 2297)
    Memory: 180.0K
       CPU: 1ms
    CGroup: /system.slice/packtd.service
           └─1365 /usr/local/bin/packtd
```

```
Apr 19 21:34:25 saturn2 systemd[1]: Started packtd. service - "Проверка журнала"
```

Давайте убедимся, что демон `packtd` не ограничен политиками безопасности SELinux. Для этого можно запустить `ps` с дополнительным ключом `-Z`, который выводит контекст SELinux для процессов:

```
[packt@saturn2 ~]$ ps -efZ | grep packtd | grep -v grep
system_u:system_r:unconfined_service_t:s0 root 1365    1    0 14:26 ?
00:00:00 /usr/local/bin/packtd
```

Атрибут безопасности `unconfined_service_t` свидетельствует о том, что `packtd` не ограничен политиками SELinux.

Далее мы создадим файлы политик безопасности для демона `packtd`.

Как создавать файлы политики

Чтобы настроить политику безопасности для `packtd`, нужно создать соответствующие файлы политики — в SELinux для этого служит утилита `sepolicy`. Кроме того, чтобы упаковать готовый двоичный файл политики безопасности, понадобится утилита `rpm-build`. Если этих инструментов по умолчанию нет в вашей системе, установите соответствующие пакеты с помощью такой команды:

```
sudo dnf install -y policycoreutils-devel rpm-build
```

¹ В случае создания или изменения имеющихся файлов сервисов рекомендуется уведомить об этом службу `systemd` следующей командой: `systemctl daemon-reload`. — *Примеч. ред.*

Вот как можно создать файлы политик для `packtd` (права суперпользователя не требуются):

```
[packt@saturn2 ~]$ sepolity generate --init /usr/local/bin/packtd
Created the following files:
/home/packt/packtd.te # Файл принудительного присвоения типов
/home/packt/packtd.if # Файл интерфейса
/home/packt/packtd.fc # Файл файловых контекстов
/home/packt/packtd_selinux.спес # Файл спецификации
/home/packt/packtd.sh # Установочный сценарий
```

Как видите, в вашем домашнем каталоге было создано *пять новых файлов*. Не потеряйте их, потому что они понадобятся в процессе настройки. Далее нужно перестроить системную политику так, чтобы она включала наш модуль политики для `packtd`.

Как настраивать политику безопасности

Мы настроим политику безопасности с помощью установочного сценария `packtd.sh`, который был создан на предыдущем шаге. Следующая команда требует привилегий суперпользователя, потому что она устанавливает только что созданную политику в систему:

```
sudo ./packtd.sh
```

Установка занимает некоторое время и дает такой результат (здесь мы приводим только фрагмент вывода):

```
[packt@saturn2 ~]$ sudo ./packtd.sh
Building and Loading Policy
+ make -f /usr/share/selinux/devel/Makefile packtd.pp
Compiling targeted packtd module
Creating targeted packtd.pp policy package
rm tmp/packtd.mod tmp/packtd.mod.fc
+ /usr/sbin/semodule -i packtd.pp
+ sepolity manpage -p . -d packtd_t
./packtd_selinux.8
+ /sbin/restorecon -F -R -v /usr/local/bin/packtd
```

Обратите внимание, что установочный сценарий переопределяет стандартный контекст безопасности SELinux для демона `packtd` с помощью команды `restorecon`. Теперь, когда мы установили политику безопасности, стоит проверить соответствующие разрешения.

Как проверять политику безопасности

Прежде всего, чтобы применить изменения в политике безопасности, давайте перезапустим демон `packtd`:

```
sudo systemctl restart packtd
```

Теперь процесс `packtd` должен отражать новый контекст безопасности SELinux. Убедитесь, что в этом контексте фигурирует новая метка `packtd_t`:

```
[packt@saturn2 ~]$ ps -efZ | grep packtd | grep -v grep
system_u:system_r:packt_t:s0 root          1781      1    0 17:06 ? 00:00:00
/usr/local/bin/packtd
```

Поскольку наш демон `packtd` теперь работает под управлением SELinux, мы должны увидеть соответствующие следы аудита в журнале AVC, где SELinux регистрирует активность системы. Утилита `ausearch` позволяет извлечь последние события с типом сообщений AVC, из которых видно, что у `packtd` нет разрешений на то, чтобы открывать файл `/var/log/messages` и записывать в него:

```
[packt@saturn2 ~]$ sudo ausearch -m AVC -ts recent
----
time->Wed Apr 19 22:32:06 2023
type=AVC msg=audit (1741462326.823:1275): avc: denied { write } for pid=14817
comm="packtd" name="messages" dev="dm-0" ino=25698477
scontext=system_u:system_r:packtd_t:s0 tcontext=system_u:object_r:var_log_t:s0
tclass=file permissive=1
----
time-> Wed Apr 19 22:32:06 2023
type=AVC msg=audit (1741462326.823:1276): avc: denied { open } for pid=14817
comm="packtd" path="/var/log/messages" dev="dm-0" ino=25698477
scontext=system_u:system_r:packtd_t:s0 tcontext=system_u:object_r:var_log_t:s0
tclass=file permissive=1
```

Чтобы подробнее разобраться, какие разрешения нужны `packtd`, мы передадим результаты `ausearch` в инструмент `audit2allow`, который умеет генерировать необходимые заглушки политик безопасности. Эта команда выводит поправки к макросам безопасности, которые нужно применить:

```
[packt@saturn2 ~]$ sudo ausearch -m AVC -ts recent | audit2allow -R
require {
    type packtd_t;
}
#===== packtd_t =====
logging_rw_generic_logs(packtd_t)
```

Ключ `-R (--reference)` в `audit2allow` генерирует заглушки и иногда может давать неточные или неполные результаты. В таких случаях может понадобиться несколько раз обновить, перезапустить и проверить соответствующие политики безопасности. Давайте внесем изменения, которые показаны в предыдущем листинге. Отредактируем файл *принудительного присвоения типов* `packtd.te`, который мы создали ранее, и добавим в него в точности те строки, которые предлагает `audit2allow`. Вот выдержка из этого файла, где выделены строки, которые нужно добавить:

```
files_read_etc_files(packtd_t)

miscfiles_read_localization(packtd_t)

require {
    type packtd_t;
```

```
}
#===== packtd_t =====
logging_rw_generic_logs(packtd_t)
```

После того как вы сохраните этот файл, нужно будет перестроить политику безопасности, перезапустить демон `packtd` и проверить журналы аудита. Для этого повторите шаги, которые мы уже выполняли:

```
sudo ./packtd.sh
sudo systemctl restart packtd
sudo ausearch -m AVC -ts recent | audit2allow -R
<no matches>
Nothing to do
```

Как видите, аудит больше не предлагает исправлений. Впрочем, иногда утилите `ausearch` может потребоваться некоторое время, чтобы обновить свой буфер недавних сообщений. Чтобы не ждать, можно задать начальную временную метку — например, любой момент после того, как вы обновили политики безопасности:

```
sudo ausearch --start 04/19/2023 '23:30:00' | audit2allow -R
```

Теперь, когда мы создали собственную политику безопасности SELinux, давайте разберемся, как ею можно управлять.

Как управлять политиками SELinux

В SELinux есть несколько утилит для того, чтобы управлять политиками и модулями безопасности. Про некоторые из этих утилит мы кратко поговорим в разделе «Как устранять неполадки SELinux», а сейчас подробнее остановимся на `semanage` и посмотрим, как с ее помощью решать некоторые задачи управления политиками безопасности.

Общий синтаксис команды `semanage` таков:

```
semanage целевой_объект [ключи]
```

Здесь *целевой_объект* обычно обозначает то или иное пространство имен, для которого определяется политика (например, `login`, `user`, `port`, `fcontext`, `boolean` или `permissive`). Давайте рассмотрим несколько примеров, чтобы получить представление о том, как работает `semanage`.

Как защитить привязку к пользовательскому порту

Предположим, что мы хотим настроить политику SELinux так, чтобы она защищала привязку службы SSH к нестандартному порту, помимо стандартного 22. С помощью следующей команды можно получить текущие записи безопасности (метки) для порта SSH:

```
[packt@saturn2 ~]$ sudo semanage port -l | grep ssh
ssh_port_t          tcp          22
```

Чтобы включить SSH на другом порте (например, 2222), нужно настроить соответствующую службу (`sshd`) так, чтобы она прослушивала этот порт. Как это сделать, показано в главе 13, а здесь мы не будем вдаваться в подробности и предположим, что служба уже привязана к порту, а нам остается только включить политику SELinux для этой привязки:

```
sudo semanage port -a -t ssh_port_t -p tcp 2222
```

Вот что делают ключи этой команды:

-a (--add)	Добавить новую запись (метку) для заданного типа (в данном случае <code>port</code>)
-t <code>ssh_port_t</code>	Тип объекта в номенклатуре SELinux
-p <code>tcp</code>	Сетевой протокол, который связан с портом

После того как эта команда выполнится, обновленная политика безопасности для типа `ssh_port_t` будет выглядеть так:

```
[packt@saturn2 ~]$ sudo semanage port -l | grep ssh
ssh_port_t                tcp                2222, 22
```

Можно удалить старую метку безопасности для порта 22, но это ни на что не повлияет, если мы отключим привязку к этому порту. Чтобы все-таки удалить соответствующую запись, можно запустить такую команду:

```
sudo semanage port -d -p tcp 22
```

За удаление метки здесь отвечает ключ `-d (--delete)`. Чтобы просмотреть наши специальные настройки для политик `semanage port`, можно использовать ключ `-C (--locallist)`:

```
sudo semanage port -l -C
```

Чтобы получить дополнительную информацию о `semanage port`, можно обратиться к соответствующей справочной странице (`man semanage port`). Далее давайте рассмотрим, как изменить разрешения безопасности для серверных приложений.

Как редактировать разрешения безопасности для целевых служб

Чтобы включать и отключать те или иные свойства целевых служб, `semanage` использует пространство имен `boolean`. Целевая служба — это демон со встроенной защитой SELinux. Для примера рассмотрим, как включить соединения FTP поверх HTTP; по умолчанию эта функция безопасности Apache (`httpd`) отключена. (Установка сервера `httpd` описана в главе 13.) Запросим соответствующие политики безопасности `httpd` с помощью уже знакомой команды:


```
[packt@saturn2 ~]$ sudo semanage boolean -l | grep httpd | grep ftp
httpd_can_connect_ftp      (off , off) Allow httpd to act as a FTP client
connecting to the ftp port and ephemeral ports
httpd_enable_ftp_server    (off , off) Allow httpd to act as a FTP server
by listening on the ftp port.
```

Как видите, соответствующая функция `httpd_enable_ftp_server` в настоящий момент отключена (первое значение `off`), и по умолчанию она тоже отключена (второе значение `off`). Ее можно включить с помощью такой команды:

```
sudo semanage boolean -m --on httpd_enable_ftp_server
```

В этой команде ключ `-m (--modify)` позволяет задать нужное значение для функции `httpd_enable_ftp_server`.

Чтобы просмотреть локальные настройки политик `semanage boolean`, укажите ключ `-C (--locallist)`:

```
[packt@saturn2 ~]$ sudo semanage boolean -l -C
httpd_enable_ftp_server    (on , on) Allow httpd to act as a FTP server
by listening on the ftp port.
```

Чтобы получить дополнительную информацию о `semanage boolean`, можно обратиться к соответствующей справочной странице (`man semanage boolean`). Теперь давайте узнаем, как изменить контекст безопасности серверных приложений.

Как редактировать контексты безопасности для целевых служб

В этом примере мы хотим обезопасить ключи SSH, которые хранятся в особом месте локальной системы — допустим, в каталоге `/etc/ssh/keys`. Поскольку соответствующая политика безопасности будет связана с файловой системой, мы задействуем команду `semanage` с пространством имен `fcontext` (файловый контекст).

Эта команда запрашивает настройки безопасности файлового контекста для служб `sshd`:

```
[packt@saturn2 ~]$ sudo semanage fcontext -l | grep sshd
```

Результат:

<code>/etc/rc.d/init.d/sshd</code>	regular file	<code>system_u:object_r:sshd_initrc_exec_t:s0</code>
<code>/etc/ssh/primers</code>	regular file	<code>system_u:object_r:sshd_key_t:s0</code>
<code>/etc/ssh/ssh_host.*_key</code>	regular file	<code>system_u:object_r:sshd_key_t:s0</code>
<code>/etc/ssh/ssh_host.*_key.pub</code>	regular file	<code>system_u:object_r:sshd_key_t:s0</code>
<code>/run/sshd\.\init\.\pid</code>	regular file	<code>system_u:object_r:sshd_var_run_t:s0</code>
<code>/run/sshd\.\pid</code>	regular file	<code>system_u:object_r:sshd_var_run_t:s0</code>
<code>/usr/bin/gssshd</code>	regular file	<code>system_u:object_r:sshd_exec_t:s0</code>
<code>/usr/bin/sshd</code>	regular file	<code>system_u:object_r:sshd_exec_t:s0</code>
<code>/usr/bin/sshd-keygen</code>	regular file	<code>system_u:object_r:sshd_keygen_exec_t:s0</code>
<code>/usr/lib/systemd/system/sshd-keygen.*</code>	regular file	<code>system_u:object_r:sshd_keygen_unit_file_t:s0</code>
<code>/usr/lib/systemd/system/sshd.*</code>	regular file	<code>system_u:object_r:sshd_unit_file_t:s0</code>
<code>/usr/libexec/openssh/sshd-keygen</code>	regular file	<code>system_u:object_r:sshd_keygen_exec_t:s0</code>
<code>/var/empty/sshd/etc/localtime</code>	regular file	<code>system_u:object_r:locale_t:s0</code>

Следующая команда добавляет наш каталог `/etc/ssh/keys/` в список целевых объектов, которые связаны с типом контекста `sshd_key_t`:

```
sudo semanage fcontext -a -t sshd_key_t '/etc/ssh/keys(/.*)?'
```

Регулярное выражение `'/etc/ssh/keys(/.*)?'` соответствует любым файлам в каталоге `/etc/ssh/keys/` и во всех его вложенных подкаталогах. С помощью ключа `-C` (`--locallist`) можно просмотреть локальные настройки политик `semanage fcontext`:

```
[packt @saturn2 ~]$ sudo semanage fcontext -l -C
SELinux fcontext                                type          Context
/etc/ssh/keys(/.*)?                             all files     system_u:object_r:sshd_key_t:s0
```

Также нужно инициализировать контекст безопасности файловой системы для каталога `/etc/ssh/keys`. Эта команда сработает, если каталог уже создан; иначе мы получим сообщение об ошибке:

```
sudo restorecon -r /etc/ssh/keys
```

Утилита SELinux `restorecon` восстанавливает контекст безопасности по умолчанию для объекта файловой системы. С ключом `-r` (или `-R`) она рекурсивно обрабатывает все объекты в указанном каталоге.

Чтобы получить дополнительную информацию о `semanage fcontext`, можно обратиться к соответствующей справочной странице (`man semanage fcontext`). Далее мы узнаем, как включить уведомительный режим для серверных приложений.

Как включить уведомительный режим для целевых служб

Ранее в этой главе мы создали пользовательский демон (`packtd`) со своей политикой безопасности. Все это время мы могли запускать `packtd` и экспериментировать с ним так, чтобы SELinux не отключал демон из-за того, что он не соответствует требованиям безопасности, несмотря на то что по умолчанию SELinux работает в принудительном (**Enforcing**) режиме. (Дополнительную информацию о принудительном и уведомительном режимах см. в разделе «Режимы SELinux».)

Дело в том, что по умолчанию SELinux применяет уведомительный режим ко всем *нецелевым* типам в системе: это такие типы (то есть домены), которые не были специально переведены в принудительный режим.

Когда мы создавали политику безопасности для демона `packtd`, мы позволили соответствующим инструментам SELinux по умолчанию сгенерировать файл принудительного присвоения типов (`packt.te`) и другие ресурсы для нашего домена. Если просмотреть этот файл, мы увидим, что наш тип `packtd_t` является уведомительным (**permissive**). Вот соответствующий отрывок из файла:

```
type packtd_t;  
type packtd_exec_t;  
init_daemon_domain(packtd_t, packtd_exec_t)  
  
permissive packtd_t;
```

Таким образом, домен `packtd_t` — уведомительный по своей природе. Чтобы ограничить разрешения `packtd`, можно удалить строку `permissive` из файла `packtd.te` и перенастроить соответствующую политику безопасности. Попробуйте заняться этим в качестве самостоятельного упражнения. Мы привели этот пример для того, чтобы далее продемонстрировать, как с помощью команды `semanage` выявить домен, который, возможно, неправильно сконфигурирован в уведомительном режиме.

Чтобы управлять уведомительным режимом для отдельных целевых объектов, команду `semanage` можно запускать с пространством имен `permissive`. Следующая команда выводит список всех доменов (типов), которые находятся в режиме `permissive`; приведем фрагмент этого списка:

```
sudo semanage permissive -l
```

Builtin Permissive Types

```
samba_bgqd_t  
systemd_pcrlock_t  
coreos_installer_generator_t  
packtd_t  
coreos_sulogin_force_generator_t
```

Можно убедиться, что в списке фигурирует наш встроенный домен `packtd_t`, который относится к уведомительным.

Та же команда `semanage permissive` позволяет временно перевести ограниченный домен в уведомительный режим — в частности, если нужно тестировать систему или устранять неполадки в той или иной функциональности. Например, следующая команда переводит демон Apache (`httpd`) в режим `Permissive`:

```
sudo semanage permissive -a httpd_t
```

После этого к списку уведомительных типов добавится специально настроенный тип:

Customized Permissive Types

```
httpd_t
```

Домены или типы, которые переведены в уведомительный режим с помощью команды `semanage permissive`, отображаются под заголовком `Customized Permissive Types`¹.

¹ Пользовательские уведомительные типы. — *Примеч. пер.*

Чтобы вернуть домен `httpd_t` в принудительный режим, можно вызвать команду `semanage permissive` с ключом `-d` (`--delete`):

```
sudo semanage permissive -d httpd_t
[packt@saturn2 ~]$ sudo semanage permissive -d httpd_t
libsemanage.semanage_direct_remove_key: Removing last permissive_httpd_t module
(no other permissive_httpd_t module exists at another priority).1
```

Обратите внимание, что с помощью команды `semanage` нельзя ограничить встроенные уведомительные типы. Как мы уже говорили, домен `packtd_t` является уведомительным по своей природе, и его нельзя перевести в принудительный режим.

Итак, теперь мы в целом представляем себе, как устроены политики безопасности SELinux. Давайте перейдем к операциям более высокого уровня, с помощью которых можно управлять SELinux в повседневных задачах администрирования.

Как устранять неполадки SELinux

Наша обзорная экскурсия по возможностям SELinux близится к завершению. В этом разделе мы познакомились с несколькими инструментами и приемами, которые позволяют исследовать конфигурацию политик безопасности и контроля доступа между субъектами (пользователями и процессами) и объектами (файлами). Неполадки SELinux обычно сводятся к тому, что система отказывает в действиях либо между теми или иными субъектами, либо между субъектом и какими-либо объектами. Эти неполадки не всегда легко выявить или устранить, но если знать, какие инструменты могут помочь, — это уже будет хорошей отправной точкой.

Вот некоторые из этих инструментов:

<code>/var/log/messages</code>	Файл журнала с трассировкой контроля доступа и отчетом о нарушениях политик SELinux ²
<code>audit2allow</code>	Генерирует разрешительные правила политики SELinux на основании трассировки журнала, соответствующей запрещенным операциям
<code>audit2why</code>	Синоним для <code>audit2allow -w</code> ; преобразовывает сообщения аудита SELinux о нарушениях политики в расширенный человекочитаемый формат
<code>ausearch</code>	Ищет записи журнала SELinux по заданным параметрам

¹ Удаляется последний модуль типа `permissive_httpd_t` (не существует других модулей этого типа с другим приоритетом). — *Примеч. пер.*

² Журнал SELinux может находиться в разных местах. Например, если запущен демон `auditd` (как происходит, например, по умолчанию в современных версиях Fedora), то журнал ведется в файле `/var/log/audit/audit.log`. — *Примеч. ред.*

<code>ls -Z</code>	Расширение команды <code>ls</code> ; выводит список объектов файловой системы с соответствующими контекстами SELinux
<code>ps -Z</code>	Расширение команды <code>ps</code> ; выводит список процессов с соответствующими контекстами SELinux
<code>restorecon</code>	Восстанавливает контексты SELinux по умолчанию для объектов файловой системы
<code>seinfo</code>	Ищет сведения в файлах политик безопасности SELinux по заданным параметрам
<code>semanage</code>	Управляет политиками SELinux
<code>semodule</code>	Управляет модулями политик SELinux
<code>sepolicy</code>	Анализирует политики SELinux
<code>sesearch</code>	Запрашивает информацию из базы данных политик SELinux

У большинства этих инструментов есть собственная справочная страница (например, `man sesearch`), где подробно описано, как использовать соответствующие команды. Кроме того, в вашем распоряжении также есть обширная документация по SELinux, и сейчас мы расскажем, как к ней обратиться.

Документация по SELinux

Богатая документация по SELinux доступна по адресу docs.redhat.com/en/documentation/red_hat_enterprise_linux/9/html/using_selinux (для RHEL 9).

Документацию по политикам SELinux также можно установить локально с помощью такой команды:

```
sudo dnf install -y selinux-policy-doc.noarch
```

Чтобы найти справочные страницы SELinux, в описании которых есть то или иное ключевое слово, можно запустить такую команду (например, для слова `report`):

```
man -k selinux | grep report
```

SELinux — одна из самых известных систем безопасности на уровне ядра Linux с богатыми возможностями конфигурации. Однако эти богатые возможности сопровождаются сложной настройкой, которая может оказаться не под силу неопытному администратору. Иногда даже квалифицированным администраторам непросто решить, какой дистрибутив Linux выбрать с учетом базового модуля безопасности. SELinux в основном доступен на платформах RHEL/Fedora, а также в Android от Google, но его можно установить и в **SUSE Linux Enterprise (SLE)**, и в Debian. Более легкую, но не менее эффективную систему безопасности предлагает **AppArmor**, который доступен по умолчанию в Ubuntu, Debian и openSUSE. Мы рассмотрим его в следующем разделе.

Модуль безопасности AppArmor

AppArmor — это модуль безопасности Linux на основе модели MAC, который ограничивает приложения определенным набором ресурсов. Механизм контроля доступа AppArmor опирается на профили безопасности, которые загружаются в ядро Linux. Каждый профиль содержит набор правил доступа к различным системным ресурсам. AppArmor можно настроить так, чтобы он принудительно контролировал доступ или чтобы просто уведомлял, когда правила нарушаются.

AppArmor проактивно защищает приложения и ресурсы операционной системы от внутренних и внешних угроз, включая атаки «нулевого дня», что позволяет предотвращать использование известных и неизвестных уязвимостей.

AppArmor встроен в основное ядро Linux, начиная с версии 2.6.36, и сейчас поставляется с Ubuntu, Debian, openSUSE и другими дистрибутивами этого семейства.

В следующих разделах мы продемонстрируем возможности AppArmor на практических примерах в среде Ubuntu Server 22.04 LTS. Большинство соответствующих утилит командной строки будут работать аналогично в любой системе, где установлен AppArmor.

Как работать с AppArmor

Утилиты командной строки AppArmor обычно требуют привилегий суперпользователя. Команда `aa-status` (или `apparmor_status`) проверяет текущее состояние AppArmor. Вот фрагмент того, что она выводит:

```
apparmor module is loaded.
136 profiles are loaded.
39 profiles are in enforce mode.
  /snap/snapd/23545/usr/lib/snapd/snap-confine
  /snap/snapd/23545/usr/lib/snapd/snap-confine//mount-namespace-capture-helper
...
92 profiles are in unconfined mode.
  1password
  Discord
...
0 processes are in kill mode.
0 processes are unconfined but have a profile defined.
0 processes are in mixed mode.
```

Здесь перечисляются профили AppArmor, которые загружены в данный момент. Поговорим о них подробнее.

Профили AppArmor

В AppArmor процессы ограничиваются с помощью профилей, которые загружаются при старте системы и работают в одном из двух режимов.

enforce	Принудительный	AppArmor не позволяет приложениям выполнять действия, на которые наложены ограничения. Попытки нарушений регистрируются в журнале <code>syslog</code> . По умолчанию Ubuntu загружает профили приложений в режиме <code>enforce</code>
complain	Уведомительный	Приложения могут выполнять запрещенные действия, а AppArmor регистрирует нарушения в журнале. Режим <code>complain</code> идеально подходит для того, чтобы тестировать профили AppArmor: потенциальные ошибки или нарушения доступа можно выявить и исправить до того, как переключать профили в режим <code>enforce</code>

А теперь давайте создадим простое приложение с профилем AppArmor.

Создание профиля

В этом разделе мы создадим простое приложение под защитой AppArmor. Надеемся, что это упражнение поможет вам в общих чертах разобраться, как работает AppArmor. Назовем наше приложение `appackt`: пусть это будет простой сценарий, который создает файл, записывает в него данные, а затем удаляет его. Мы хотим добиться, чтобы AppArmor не позволял приложению обращаться к любым другим путям в локальной системе. Рассматривайте этот сценарий как тривиальную форму переработки журналов. Вот его код:

```
#!/bin/bash
# Каталог ./log должен существовать
if [[ ! -d "./log" ]] then
    echo "Нет каталога журнала!"
    exit 1
fi
LOG_FILE="./log/appackt"
echo "Создается ${LOG_FILE}..."
touch ${LOG_FILE}
echo "Запись в ${LOG_FILE}..."
date +"%b %d %T ${HOSTNAME}: Привет от Packt! 🐙" >> ${LOG_FILE}
echo "Чтение из ${LOG_FILE}..."
cat ${LOG_FILE}
echo "Удаляется ${LOG_FILE}..."
rm ${LOG_FILE}
```

Чтобы сценарий работал, нужно создать каталог `log` в том же месте, что и сам сценарий:

```
mkdir ./log
```

Давайте сделаем сценарий исполняемым и запустим его. Вот что он выводит:

```
packt@neptune:~$ chmod a+x appackt
packt@neptune:~$ ./appackt
Создается ./log/appackt...
Запись в ./log/appackt...
```

```
Чтение из ./log/appackt...
Mar 10 17:00:02 neptune: Привет от Packt! 🐙
Удаляется ./log/appackt...
```

Теперь давайте наладим защиту нашего сценария с помощью AppArmor. Прежде всего нужно установить пакет `apparmor-utils` — **набор инструментов AppArmor**:

```
sudo apt install -y apparmor-utils
```

Нам понадобятся два инструмента: `aa-genprof`, который генерирует профили безопасности AppArmor, и `aa-logprof`, который их обновляет.

Мы запустим `aa-genprof` так, чтобы он отслеживал наше приложение во время выполнения и собирал сведения о нем для AppArmor. Во время этой процедуры он несколько раз предложит выбрать, какие действия должен предпринимать AppArmor в тех или иных обстоятельствах.

После того как профиль будет готов, мы используем утилиту `aa-logprof`, чтобы внести дальнейшие коррективы во время тестирования в уведомительном режиме, если возникнут какие-либо нарушения.

Начнем с `aa-genprof`. Нам понадобятся два терминала: в *терминале 1* мы запустим сеанс мониторинга `aa-genprof`, а в *терминале 2* — наш сценарий¹.

- Для начала запустим в *терминале 1* такую команду:

```
sudo aa-genprof ./appackt
```

Она выведет справочную информацию об утилите `aa-genprof` и профилях AppArmor, а также первый запрос:

```
packt@neptune:~$ sudo aa-genprof ./appackt
Updating AppArmor profiles in /etc/apparmor.d.
Before you begin, you may wish to check if a profile already exists for the
application you wish to confine. See the following wiki page for more
information:
https://gitlab.com/apparmor/apparmor/wikis/Profiles
Profiling: /home/neptune/appackt
Please start the application to be profiled in another window and exercise
its functionality now.
Once completed, select the "Scan" option below in order to scan the system
logs for AppArmor events.
For each AppArmor event, you will be given the opportunity to choose whether
the access should be allowed or denied.
```

¹ Чтобы переключаться из одного виртуального терминала в другой, можно использовать сочетания клавиш `Ctrl + Alt + Fn` (см. главу 1). Например, здесь можно запустить `aa-genprof` в основном терминале (`tty1`), а затем с помощью `Ctrl + Alt + F2` переключиться в `tty2` и запустить там сценарий `appackt`. Чтобы вернуться в основной терминал, нажмите `Ctrl + Alt + F1`. Чтобы проверить, в каком терминале вы находитесь, запустите команду `tty`. — *Примеч. ред.*


```
[(S)can system log for AppArmor events] / (F)inish
# S — сканировать системный журнал в поисках событий AppArmor
# F — завершить
```

- Пока не отвечайте на запрос, а переключитесь в *терминал 2* и запустите сценарий `appackt`:

```
./appackt
```

- Теперь вернитесь в *терминал 1* и ответьте на запрос `aa-genprof`: нажмите `S`, чтобы просканировать системный журнал в поисках событий, которые представляют интерес для AppArmor.
- После сканирования система предложит вам задать нужные разрешения и действия для разных событий. Сначала понадобится разрешить выполнение команд, которые используются в сценарии, — `touch`, `date`, `cat` и `rm`. Например, так выглядит запрос для `touch`:

```
Reading log entries from /var/log/syslog.
```

```
Profile: /home/neptune/appackt
Exerute: /usr/bin/touch
Severity: 3
```

```
(I)nherit / (C)hild / (P)rofile / (N)amed / (X) ix On / (D)eny / Abo(r)t /
(F)inish
```

Во всех таких запросах нажмите `I` (`Inherit`), чтобы дочерние процессы (например, `touch`) наследовали разрешения и ограничения родительского процесса (`appackt`).

- Затем последует еще одна серия запросов: система предложит настроить права доступа к файлам, к которым обращается `appackt`. Например, так выглядит запрос для файла `/home/packt/log/appackt`, с которым работает наш сценарий:

```
Complain-mode changes:
```

```
Profile: /home/packt/appackt
Path:    /home/packt/log/appackt
New Mode: owner w
Severity: 6
```

```
[1 - owner /home/*/log/appackt w,]
2 - owner /home/packt/log/appackt w,
(A)llow / [(D)eny] / (I)gnore / (G)lob / Glob with (E)xtension / (N)ew /
Audi(t) / (O)wner permissions off / Abo(r)t / (F)inish
```

Вы увидите похожие запросы и для других ресурсов — например, терминала (`/dev/tty`) и кэша общих библиотек (`/etc/ld.so.cache`). Каждый раз нажимайте `A` (`Allow`), чтобы разрешить доступ.

- Наконец, `aa-genprof` сообщит о том, что профилирование закончено, и предложит сохранить изменения или выполнить другие действия:

```
== Changed Local Profiles ==
```

```
The following local profiles were changed. Would you like to save them?
```

```
[1 - /home/packt/appackt]
(S)ave Changes / Save Selec(t)ed Profile / [(V)iew Changes] / View Changes
b/w (C)lean profiles / Abo(r)t
```

Нажмите *S* (Save Changes), чтобы сохранить новый профиль. Все профили приложений хранятся в каталоге `/etc/apparmor.d`, и после нашего профилирования там появится новый файл `home.packt.appackt`.

- После этого `aa-genprof` вернется в начало и предложит снова просканировать системный журнал или завершить работу. Нажмите *F* (Finish), чтобы выйти из утилиты.

Теперь AppArmor по умолчанию защищает наше приложение (`appackt`) в принудительном режиме (`enforce`).

Для дальнейших шагов нам понадобится только один терминал. Давайте запустим команду `aa-logprof`, чтобы дополнительно настроить профиль безопасности для `appackt` (если понадобится):

```
sudo aa-logprof
```

Вы снова увидите несколько запросов, похожих на предыдущие, которые уточняют разрешения для нашего сценария или других приложений. Отвечайте на запросы *I* (наследовать) и *A* (разрешить) там, где это уместно, — здесь мы не будем вдаваться в дальнейшие подробности, потому что в предыдущем упражнении вы уже познакомились с этими запросами и вариантами ответов. Однако всегда имеет смысл задуматься о том, как выбрать наиболее подходящие разрешения.

Возможно, команду `aa-logprof` придется запустить несколько раз, потому что на каждой итерации могут обнаруживаться новые разрешения в зависимости от дочерних процессов, которые породил наш сценарий. Рано или поздно `aa-logprof` перестанет находить новые конфликты, и сценарий `appackt` успешно запустится.

В результате этого процесса в базе данных AppArmor может оказаться несколько бесхозных записей, которые остались от предыдущих попыток настроить защиту. Названия всех этих записей начинаются с пути к приложению `appackt`, например:

```
4 profiles are in complain mode.
/home/packt/appackt/null-/usr/bin/cat
/home/packt/appackt/null-/usr/bin/date
/home/packt/appackt/null-/usr/bin/rm
/home/packt/appackt/null-/usr/bin/touch
```

Эти записи можно очистить с помощью такой команды:

```
sudo aa-remove-unknown
```

Когда все готово, запустите такую команду, чтобы перевести профиль `appackt` в принудительный режим (`enforce`):

```
sudo aa-enforce /etc/apparmor.d/home.packkt.appackt
```

Теперь можно убедиться, что наше приложение действительно находится под защитой AppArmor:

```
sudo aa-status
```

Вот соответствующий фрагмент отчета:

```
apparmor module is loaded.  
33 profiles are loaded.  
33 profiles are in enforce mode.  
    /home/packt/appackt
```

Как и ожидалось, наше приложение отображается в режиме `enforce`.

Теперь давайте проверим, что на приложение действительно распространяются соответствующие политики безопасности AppArmor. Отредактируйте сценарий `appackt` и укажите в нем путь к другому файлу журнала:

```
LOG_FILE="./log/appackt.log"
```

Если теперь запустить приложение, мы увидим сообщения об ошибках, которые говорят о том, что `appackt` пытается получить доступ к ресурсам вне разрешенных границ AppArmor:

```
packt@neptune:~$ ./appackt  
Создается файл ./log/appackt.log...  
touch: cannot touch './log/appackt.log': Permission denied  
Запись в файл ./log/appackt.log...  
./appackt: line 12: ./log/appackt.log: Permission denied  
Чтение из файла ./log/appackt.log...  
cat: ./log/appackt.log: No such file or directory  
Удаляется файл ./log/appackt.log...  
rm: cannot remove './log/appackt.log': No such file or directory
```

Если бы было нужно внести дополнительные изменения в наше приложение и перетестировать его заново, нам понадобилось бы изменить режим профиля на уведомительный (`complain`), а затем повторить профилирование, как описывалось ранее в этом разделе. Следующая команда переводит профиль приложения в уведомительный режим:

```
sudo aa-complain /home/packt/appackt
```

Профили AppArmor — это обычные текстовые файлы, которые хранятся в каталоге `/etc/apparmor.d/`. Чтобы создавать или изменять профили, можно вручную редактировать соответствующие файлы или запускать инструменты `aa-genprof` и `aa-logprof`, как описывалось в этом разделе.

Далее рассмотрим, как отключить или включить профили приложений AppArmor.

Как отключать и включать профили

Если профиль приложения вызывает проблемы, иногда его имеет смысл отключить, чтобы заменить на улучшенную версию. Посмотрим, как это сделать.

Сначала найдем файл с нужным профилем: он находится в каталоге `/etc/apparmor.d/` и называется так же, как полный путь к файлу приложения, но с точками (.) вместо косых черт (/). Наше приложение `appackt` находится в каталоге `/home/packt`, а значит, ему соответствует файл профиля `/etc/apparmor.d/home.packt.appackt`. В этом можно убедиться с помощью `ls`:

```
packt@neptune:~$ ls -al /etc/apparmor.d/home.packt.appackt
-rw----- 1 root root 478 Mar 10 17:36 /etc/apparmor.d/home.packt.appackt
```

Чтобы отключить профиль, нужно запустить такие команды:

```
sudo ln -s /etc/apparmor.d/home.packt.appackt /etc/apparmor.d/disable/
sudo apparmor_parser -R /etc/apparmor.d/home.packt.appackt
```

Если после этого запустить `aa-status`, в списке больше не будет нашего профиля `appackt`. Однако на него ведет символическая ссылка `/etc/apparmor.d/disable/home.packt.appackt`.

В этом режиме AppArmor не накладывает никаких ограничений на сценарий `appackt`. Чтобы снова включить соответствующий профиль безопасности, можно выполнить такие команды:

```
sudo rm /etc/apparmor.d/disable/home.packt.appackt
sudo apparmor_parser -r /etc/apparmor.d/home.packt.appackt
```

Теперь `aa-status` снова отображает профиль `appackt` в принудительном режиме (`enforce`).

В этих процедурах мы использовали утилиту `apparmor_parser`, которая с ключом `-r (--replace)` загружает профили безопасности в ядро, а с ключом `-R (--remove)` выгружает их оттуда.

Если просто удалить профиль безопасности AppArmor из файловой системы, результат будет таким же, как если бы мы его отключили. Если мы удаляем профиль, перед этим не выгрузив его из ядра (с помощью `apparmor_parser -R`), то оставшиеся бесхозные записи можно очистить с помощью команды `aa-remove-unknown`.

Давайте подытожим наш относительно краткий разговор об AppArmor некоторыми заключительными соображениями.

Заключительные соображения

Работать с AppArmor проще, чем с SELinux, особенно когда нужно создавать политики безопасности или переключаться между принудительным и уведомительным режимами. SELinux переключает разрешительный контекст только

для всей системы, а AppArmor умеет делать это на уровне приложений. С другой стороны, не всегда есть возможность выбирать между модулями безопасности, потому что некоторые из ведущих дистрибутивов Linux поддерживают только одну из двух систем. AppArmor используется в Debian, Ubuntu и openSUSE, в то время как SELinux — в RHEL/Fedora и SLE. Теоретически всегда можно попытаться перенести соответствующие модули ядра из одного дистрибутива в другой, но это нетривиальная задача.

В заключение еще раз подчеркнем, что в общей картине безопасности и SELinux, и AppArmor относятся к системам управления доступом, которые действуют локально в системе. Когда речь заходит о том, чтобы защищать приложения и компьютерные системы от внешнего мира, в дело вступают межсетевые экраны, о которых мы поговорим в следующем разделе.

Межсетевые экраны в Linux

Под межсетевым экраном (брандмауэром) обычно понимают службу сетевой безопасности, которая размещается между двумя сетями, чтобы отслеживать трафик и контролировать доступ к этим сетям. Чаще всего межсетевой экран защищает локальную сеть от нежелательного вторжения или внешних атак, но он также может блокировать нежелательный трафик, который исходит из локальной сети в общедоступный интернет. Технически межсетевой экран пропускает или блокирует входящий и исходящий сетевой трафик на основе определенных правил безопасности.

Например, экран может блокировать все входящие сетевые протоколы, кроме избранных (таких, как SSH и HTTP/HTTPS). Он также может запрещать всем узлам локальной сети, кроме избранных, устанавливать исходящие соединения определенного типа: например, не допускать соединений **SMTP** ни от каких узлов, кроме локальных почтовых серверов.

На рис. 9.1 показано простое развертывание межсетевого экрана, который регулирует трафик между локальной сетью и интернетом.

Исходящие правила безопасности не позволяют взломанным компьютерам или недобросовестным пользователям направлять атаки в публичный интернет. Это приносит пользу внешним сетям, но в конечном итоге такая защита важна и для организации, иначе интернет-провайдеры могут наложить ограничения на ее сеть за недоброкачественный интернет-трафик.

Наладка межсетевого экрана обычно заключается в том, что создается глобальная политика безопасности по умолчанию, а затем настраиваются конкретные исключения из общих правил; исключения основаны на номерах портов (протоколах), IP-адресах и других критериях.

В следующих разделах мы рассмотрим различные реализации межсетевых экранов и системы управления ими. Сначала давайте вкратце познакомимся с тем, как межсетевой экран отслеживает и контролирует сетевой трафик.

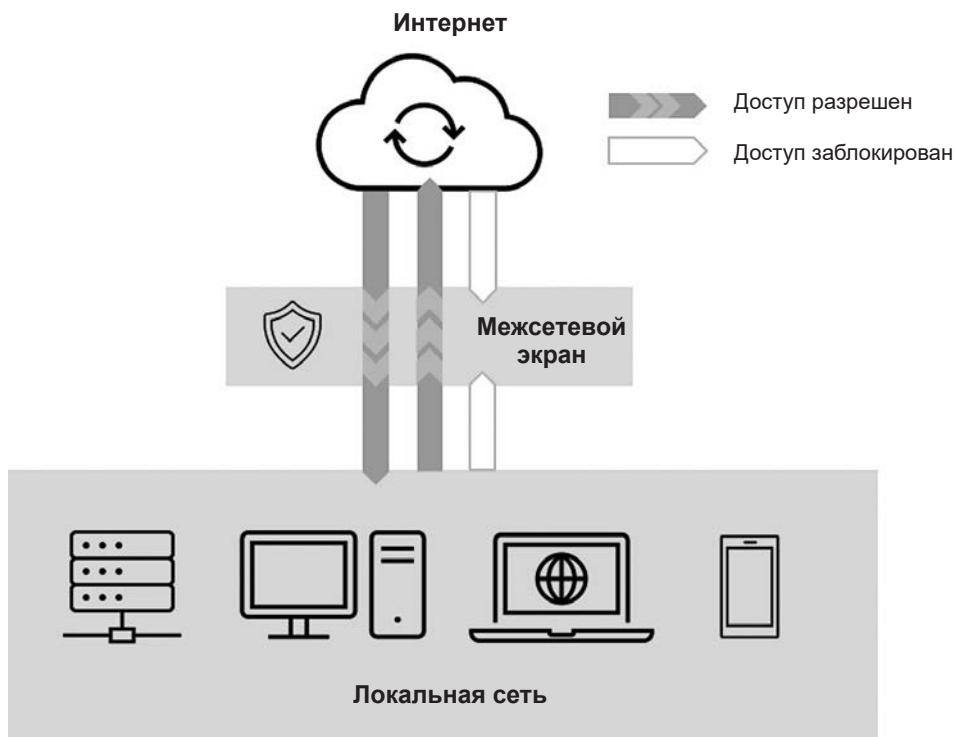


Рис. 9.1. Простая схема межсетевого экрана

Цепочки в межсетевых экранах

На высоком уровне стек TCP/IP в ядре Linux обычно выполняет такие операции:

- принимает данные от приложений (процессов) и сериализует их в сетевые пакеты, которые направляет получателю (в общем случае — другому узлу в сети), идентифицируя его по IP-адресу и порту;
- принимает данные из сети, десериализует сетевые пакеты в данные приложений и доставляет их нужным процессам.

В идеале на этих этапах ядро Linux не должно изменять сетевые данные каким-либо особым образом, кроме того, чтобы формировать их в соответствии с протоколами TCP/IP. Однако в распределенных сетевых средах, которые могут быть небезопасными, данные часто стоит дополнительно проверять. Ядро должно предоставлять механизмы, которые могут перехватывать пакеты данных, чтобы фильтровать и модифицировать их в зависимости от различных критериев. Именно здесь в дело вступают межсетевые экраны и другие средства сетевой безопасности и обнаружения вторжений. Они приспособляются к ин-

терфейсу на уровне ядра, который фильтрует пакеты TCP/IP, и отслеживают их прохождение. Общая схема перехвата пакетов выглядит так:

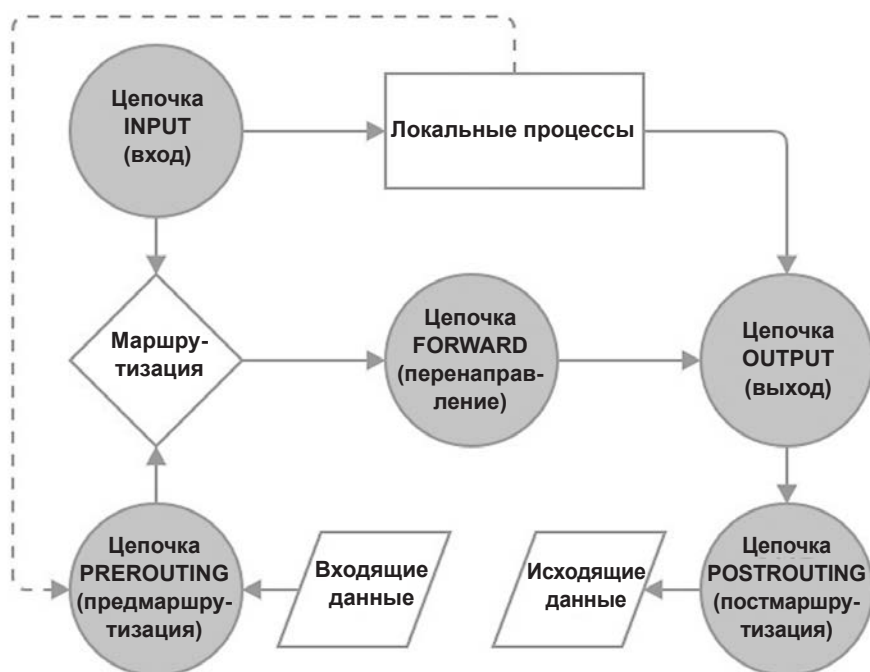


Рис. 9.2. Перехват пакетов в межсетевом экране Linux

Когда входящие данные попадают в межсетевой экран, он маршрутизирует их в зависимости от назначения пакета. В результате маршрутизации пакет может направиться либо по цепочке **INPUT** («вход», для локального узла), либо по цепочке **FORWARD** («перенаправление», для удаленного узла). Эти цепочки могут по-разному модифицировать входящие данные с помощью перехватчиков, которые реализуются средствами сетевой безопасности или межсетевыми экранами. По умолчанию ядро не изменяет пакеты, которые проходят через эти цепочки.

Цепочка **INPUT** («вход») в конечном итоге направляет пакеты в процесс **локального приложения**, которое потребляет данные. Эти локальные приложения обычно представляют собой процессы пользовательского пространства, такие как сетевые клиенты (например, веб-браузеры, клиенты SSH и почтовые программы) или сетевые серверы (например, веб-серверы и серверы электронной почты). Впрочем, к ним также могут относиться процессы пространства ядра, например сервер NFS.

И цепочка **FORWARD** («перенаправление»), и локальные процессы направляют пакеты данных в цепочку **OUTPUT** («выход»), прежде чем они попадут в сеть.

Любая из цепочек может фильтровать пакеты на основе определенных критериев, например:

- IP-адрес отправителя или получателя;
- порт отправителя или получателя;
- сетевой интерфейс, через который передаются данные.

Каждая цепочка — это набор правил безопасности, с которыми сопоставляются входящие пакеты. Если пакет подпадает под то или иное правило, ядро выполняет над пакетом **целевые операции** (или просто **цели**), которые указаны в правиле. Вот некоторые предопределенные цели:

Цель	Описание
ACCEPT	Принять пакет для дальнейшей обработки
REJECT	Отклонить пакет
DROP	Игнорировать пакет
QUEUE	Передать пакет процессу пользовательского пространства
RETURN	Прекратить обрабатывать пакет и вернуть данные в предыдущую цепочку

Полный список предопределенных целей можно найти на справочной странице `iptables`.

В следующих разделах мы рассмотрим самые распространенные механизмы и инструменты сетевой безопасности, которые относятся к сетевому стеку на уровне ядра и межсетевым экранам. Мы начнем с `Netfilter` — системы фильтрации пакетов в пространстве ядра Linux, а затем рассмотрим `iptables` — традиционный интерфейс для настройки `Netfilter` в пользовательском пространстве. `iptables` — это очень настраиваемое и гибкое решение для межсетевого экрана. Затем мы кратко расскажем о `nftables` — инструменте следующего поколения, который реализует большую часть сложных возможностей `iptables` и оборачивает их в относительно простой в использовании интерфейс командной строки. Наконец, мы немного отклонимся от внутриядерных механизмов фильтрации пакетов и рассмотрим **системы управления межсетевыми экранами**: `firewalld` (для RHEL/Fedora) и `ufw` (для Debian/Ubuntu) — два удобных фронтенда, с помощью которых можно настраивать межсетевые экраны в основных дистрибутивах Linux.

Давайте начнем с системы `Netfilter`.

Представляем Netfilter

Netfilter — это система фильтрации пакетов в ядре Linux, которая предоставляет гибко настраиваемые обработчики событий (так называемые перехватчики) для того, чтобы управлять операциями, связанными с сетью, например:

- принимать или отклонять пакеты;
- маршрутизировать и пересылать пакеты;
- преобразовывать IP-адреса (NAT/NAPT).

Приложения, реализующие фреймворк Netfilter, используют функции обратного вызова; эти функции основаны на перехватчиках, зарегистрированных в модулях ядра, которые управляют сетевым стеком. В свою очередь, функциям обратного вызова соответствуют правила безопасности и профили, которые регулируют поведение каждого пакета, проходящего через сетевые цепочки.

Приложения межсетевых экранов — наиболее характерные примеры реализации фреймворка Netfilter. Администраторам и продвинутым пользователям стоит понимать, как устроены перехватчики Netfilter, чтобы создавать надежные правила и политики межсетевого экрана. Давайте рассмотрим эти перехватчики.

Перехватчику Netfilter

По мере того как пакеты проходят по различным цепочкам сетевого стека, Netfilter возбуждает события для модулей ядра, которые зарегистрированы с помощью соответствующих **перехватчиков**¹. Модуль или приложение для фильтрации пакетов (например, межсетевой экран), который реализует эти перехватчики, оповещается о соответствующих событиях, после чего обрабатывает пакет, как предписывают имеющиеся правила.

Для приложений, которые фильтруют пакеты, доступно пять перехватчиков Netfilter. Каждый из них соответствует определенной цепочке и срабатывает, когда происходит то или иное событие:

Перехватчик	Цепочка	Описание
NF_IP_PRE_ROUTING	PREROUTING	После того как в сетевой стек поступил пакет входящего трафика, но до того, как принято решение о том, куда его маршрутизировать
NF_IP_LOCAL_IN	INPUT	После маршрутизации входящего пакета, если он адресован локальному узлу
NF_IP_FORWARD	FORWARD	После маршрутизации входящего пакета, если он адресован удаленному узлу
NF_IP_LOCAL_OUT	OUTPUT	Сразу после того, как в сетевой стек поступил пакет исходящего трафика, сформированный на локальном узле
NF_IP_POST_ROUTING	POSTROUTING	После маршрутизации исходящего или перенаправленного пакета, непосредственно перед тем, как он выходит из сетевого стека

¹ Перехватчик (*hook*, также используется калька *хук*) — это программный компонент, который реализуется на уровне ядра и позволяет перехватывать системные вызовы, прежде чем они будут выполнены в штатном режиме. — *Примеч. ред.*

Модули ядра или приложения, которые зарегистрированы на перехватчики Netfilter, должны обозначать свой приоритет, который позволяет определить, в каком порядке вызывать модули, когда сработает перехватчик. Благодаря этому механизму можно детерминированно упорядочить несколько модулей (или несколько экземпляров одного и того же модуля), которые были зарегистрированы на один и тот же перехватчик. Когда зарегистрированный модуль заканчивает обрабатывать пакет, он позволяет Netfilter решать, как дальше поступать с пакетом.

Разработкой фреймворка Netfilter занимается сообщество профессионалов, которые представляют движение за свободное и открытое программное обеспечение. Чтобы ближе познакомиться с Netfilter, можно начать с официального сайта netfilter.org.

Один из самых известных пакетов прикладных модулей для Netfilter — `iptables`. Это популярная система управления межсетевыми экранами с интерфейсом прямого доступа к механизму фильтрации пакетов Netfilter. Давайте изучим `iptables` на практике, чтобы глубже раскрыть функциональные аспекты Netfilter.

Как работать с iptables

`iptables` — это относительно низкоуровневая система управления межсетевыми экранами для Linux и утилита командной строки, которая регулирует сетевой трафик с помощью цепочек Netfilter. `iptables` работает в соответствии с **правилами**, которые группируются в **цепочки**. Каждое правило содержит критерии, по которым выбираются пакеты, проходящие через определенную цепочку. В свою очередь, правила организованы в **таблицы**, каждая из которых соответствует той или иной стратегии обработки пакетов:

Таблица	Описание	Цепочки				
		PREROUTING	INPUT	FORWARD	OUTPUT	POSTROUTING
filter	Таблица по умолчанию, на основе которой Netfilter принимает решение, разрешать ли пакетам проходить через соответствующие цепочки		☑	☑	☑	
nat	Для пакетов, у которых нужно преобразовать адреса и/или порты источника или назначения	☑			☑	☑
mangle	Позволяет изменять отдельные заголовки IP, такие как MSS или TTL	☑	☑	☑	☑	☑

Таблица	Описание	Цепочки				
		PREROUTING	INPUT	FORWARD	OUTPUT	POSTROUTING
raw	Позволяет отключать отслеживание соединений (цель NOTRACK) в определенных пакетах, в основном для того, чтобы обрабатывать данные, не сохраняя состояния, а также оптимизировать производительность	✓			✓	
security	Для принудительного управления доступом (MAC), когда пакеты подвергаются ограничениям политик SELinux		✓	✓	✓	

В сетевом стеке ядра пакеты проходят по таким цепочкам:

Входящие пакеты, адресованные локальному узлу	PREROUTING INPUT
Входящие пакеты, адресованные удаленному узлу	PREROUTING FORWARD POSTROUTING
Пакеты, которые генерируются локально прикладными процессами	OUTPUT POSTROUTING

Теперь, когда мы познакомились с некоторыми вводными понятиями, давайте рассмотрим несколько практических примеров, чтобы разобраться, как работает `iptables`.

Мы будем демонстрировать примеры в Fedora 37, но они должны работать во всех основных дистрибутивах Linux. Обратите внимание, что, начиная с RHEL 7, межсетевыми экранами по умолчанию управляет приложение `firewalld` (см. «Системы управления межсетевыми экранами» далее в этой главе). Чтобы использовать `iptables`, сначала нужно отключить `firewalld`:

```
[packt@saturn2 ~]$ sudo systemctl stop firewalld

[packt@saturn2 ~]$ sudo systemctl disable firewalld
Removed '/etc/systemd/system/multi-user.target.wants/firewalld.service'.
Removed '/etc/systemd/system/dbus-org.fedoraproject.FirewallD1.service'.

[packt@saturn2 ~]$ sudo systemctl mask firewalld
Created symlink '/etc/systemd/system/firewalld.service'
```

Затем установите пакет `iptables-services`:

```
sudo dnf install iptables-services
```

Если в вашем Ubuntu не установлен `iptables`, его можно установить с помощью команды `sudo apt install iptables`.

Теперь займемся настройкой `iptables`.

Как настраивать `iptables`

Команда `iptables` требует привилегий суперпользователя. Давайте сначала изучим текущую конфигурацию. Чтобы просмотреть правила в той или иной цепочке для определенной таблицы, используется такой синтаксис:

```
sudo iptables -L [цепочка] [-t таблица]
```

Ключ `-L (--list)` задает цепочку, а ключ `-t (--table)` — таблицу. Указывать конкретную цепочку и/или таблицу необязательно: если не указана цепочка, выводятся *все* цепочки в целевой таблице, а если не указана таблица, то по умолчанию предполагается таблица `filter`. Таким образом, следующая команда перечисляет все цепочки и соответствующие правила для таблицы `filter`:

```
[packt@saturn2 ~]$ sudo iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source                destination

Chain FORWARD (policy ACCEPT)
target     prot opt source                destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source                destination
```

Поскольку мы пока никак не настраивали межсетевой экран, здесь не отображается ни одного правила. Если бы мы хотели запросить более конкретные сведения — например, все правила цепочки `INPUT` для таблицы `nat`, — эти параметры нужно было бы указать после соответствующих ключей:

```
sudo iptables -L INPUT -t nat
```

Ключ `-t` обязателен только в том случае, если нас интересуют правила `iptables` не для таблицы `filter` по умолчанию, а для другой таблицы.

Если вы составляете правила межсетевого экрана с нуля, это обычно рекомендуется делать так:

1. Удалить все остатки текущей конфигурации межсетевого экрана.
2. Задать политику межсетевого экрана по умолчанию.
3. Создать правила межсетевого экрана, причем упорядочить их так, чтобы сначала следовали более специфичные (или ограничительные) правила.
4. Сохранить конфигурацию.

Давайте воспроизведем эти шаги на практике и создадим простую конфигурацию межсетевого экрана в таблице `filter`:

1. **Очистите текущую конфигурацию.** Запустите эти команды, чтобы стереть существующие правила из цепочек таблицы `filter` (`INPUT`, `FORWARD` и `OUTPUT`):

```
[packt@saturn2 ~]$ sudo iptables -v -F INPUT
Flushing chain `INPUT'
```

```
[packt@saturn2 ~]$ sudo iptables -v -F FORWARD
Flushing chain `FORWARD'
```

```
[packt@saturn2 ~]$ sudo iptables -v -F OUTPUT
Flushing chain `OUTPUT'
```

В этом примере мы запустили команду `iptables` с ключом `-v` (`--verbose`), который обеспечивает подробный вывод. Без этого ключа команда ничего не выводит (если не происходит ошибок).

Операция очистки удаляет все правила в указанной цепочке. Пожалуйста, будьте осторожны: очистка отключит ваш межсетевой экран, а также может заблокировать подключение по SSH, если вы его используете.

2. **Задайте политику межсетевого экрана по умолчанию.** По умолчанию `iptables` беспрепятственно пропускает по сетевому стеку все пакеты. Чтобы межсетевой экран обеспечивал надежную защиту, укажите `DROP` в качестве цели по умолчанию для каждой цепочки:

```
sudo iptables -P INPUT DROP
sudo iptables -P FORWARD DROP
sudo iptables -P OUTPUT DROP
```

Параметр `-P` (`--policy`) задает определенную цель (например, `DROP`) в качестве политики для определенной цепочки (например, `INPUT`). Цель `DROP` заставляет систему отбрасывать все пакеты.

Если бы на этом этапе мы сохранили конфигурацию межсетевого экрана, система перестала бы принимать входящие пакеты и передавать исходящие. Поэтому будьте осторожны, чтобы случайно не потерять доступ к системе, если вы подключены по SSH или если у вас нет прямого доступа к консоли.

3. **Создайте правила межсетевого экрана.** Давайте создадим несколько правил межсетевого экрана — например, чтобы он позволял принимать соединения по протоколам SSH, DNS и HTTPS.

Следующие команды предоставляют доступ по SSH из локальной сети (`192.168.0.0/24`):

```
sudo iptables -A INPUT -p tcp --dport 22 -m state \
--state NEW,ESTABLISHED -d 192.168.0.0/24 -j ACCEPT
sudo iptables -A OUTPUT -p tcp --sport 22 -m state \
--state ESTABLISHED -s 192.168.0.0/24 -j ACCEPT
```

Поясним аргументы, которые здесь используются:

Ключ		Значение	Описание
Короткая форма	Длинная форма		
-A	--append	INPUT	Цепочка, в конец которой добавляется правило
-p	--protocol	tcp	Протокол, по которому передаются пакеты
--sport	--source-port	22	Порт отправителя пакетов
--dport	--destination-port	22	Порт получателя пакетов
-m	--match	state	Расширение iptables, которое предоставляет дополнительные свойства пакетов
--state	нет	NEW, ESTABLISHED	Состояние пакета (обеспечивается расширением state)
-s	--source	192.168.0.0/24	IP-адрес и маска подсети отправителя пакетов
-d	--destination	192.168.0.0/24	IP-адрес и маска подсети получателя пакетов
-j	--jump	ACCEPT	Целевое действие (в данном случае — «принять»)

Каждая из этих двух команд создает отдельное правило, которое обеспечивает доступ по SSH. Первое правило разрешает входящий трафик SSH (--dport 22) для новых и существующих соединений (-m state --state NEW, ESTABLISHED), а второе — исходящий трафик SSH (--sport 22) для существующих соединений (--state ESTABLISHED).

Похожие команды позволяют включить трафик HTTPS:

```
sudo iptables -A INPUT -p tcp --dport 443 -m state \
  --state NEW,ESTABLISHED -j ACCEPT
sudo iptables -A OUTPUT -p tcp --sport 443 -m state \
  --state ESTABLISHED,RELATED -j ACCEPT
```

Наконец, чтобы включить трафик DNS, можно использовать такие команды:

```
sudo iptables -A INPUT -p udp --dport 53 -j ACCEPT
sudo iptables -A OUTPUT -p udp --sport 53 -j ACCEPT
```

Чтобы получить дополнительную информацию о параметрах и расширениях iptables, обратитесь к соответствующим справочным страницам:

- `man iptables`
- `man iptables-extensions`

4. **Сохраните конфигурацию.** Чтобы сохранить текущую конфигурацию `iptables`, запустите такую команду:

```
[packt@saturn2 ~]$ sudo service iptables save  
iptables: Saving firewall rules to /etc/sysconfig/iptables: [ OK ]
```

Если вы хотите сохранить текущую конфигурацию `iptables` в файл (например, `iptables.config`), чтобы использовать ее впоследствии, запустите команду `iptables-save` с ключом `-f` (`--file`):

```
sudo iptables-save -f iptables.config
```

Сохраненную конфигурацию можно позже восстановить с помощью такой команды:

```
sudo iptables-restore ./iptables.config
```

Если у вас другой путь к файлу конфигурации, укажите его вместо `./iptables.config`.

Более сложные правила и продвинутые возможности `iptables` выходят за рамки темы этой главы. Мы надеемся, что примеры, которые мы здесь привели, вкупе с теоретическим введением в `iptables` послужат вам хорошей отправной точкой для того, чтобы изучать более сложную настройку. Кроме того, в современных дистрибутивах Linux популярна система фильтрации пакетов `nftables`, а такие инструменты управления межсетевыми экранами, как `ufw` и `firewalld`, по умолчанию работают во многих дистрибутивах, включая Fedora, RHEL, openSUSE и Ubuntu.

Далее мы рассмотрим `nftables` — более новый фреймворк, который был спроектирован и разработан в рамках проекта Netfilter.

Как работать с nftables

Фреймворк `nftables`, который пришел на замену `iptables`, — это система управления межсетевыми экранами, которая поддерживает фильтрацию пакетов, NAT и различные преобразования пакетов. В `nftables` заметно улучшены функциональность, удобство и производительность, например:

- вместо линейной обработки правил используются таблицы поиска;
- правила применяются по отдельности; не требуется каждый раз обрабатывать весь набор правил;
- единая структура для протоколов IPv4 и IPv6;
- нет специфических расширений для отдельных протоколов.

`nftables` в целом опирается на те же функциональные принципы, что и `iptables`, в частности, цепочки так же хранятся в таблицах, и каждая цепочка представляет собой набор правил о том, как фильтровать пакеты.

`nftables` поставляется с дистрибутивами Debian и RHEL/Fedora как система фильтрации пакетов по умолчанию, заменяя `iptables` и другие родственные инструменты. Утилита `nft` позволяет работать с конфигурацией `nftables` из командной строки. Впрочем, некоторым специалистам удобнее использовать другой фронтенд — например, `firewalld`, который тоже поддерживает `nftables`. В RHEL/Fedora `firewalld` служит стандартной системой управления межсетевыми экранами.

В этом разделе мы покажем несколько примеров того, как с помощью `nftables` и соответствующих утилит командной строки можно выполнять простые задачи по настройке межсетевого экрана. Давайте вкратце рассмотрим подготовительные шаги, которые необходимы, чтобы выполнять упражнения в этом разделе.

Технические требования для упражнений из этого раздела

Если в вашей системе пакет `nftables` не установлен по умолчанию, его можно установить с помощью такой команды:

```
sudo yum install -y nftables
```

Во всех примерах в этом разделе используется дистрибутив Fedora 37. Если до этого вы выполняли упражнения из раздела «Как работать с `iptables`», отключите `iptables` с помощью таких команд:

```
sudo systemctl stop iptables
sudo systemctl disable iptables
```

Затем включите и запустите `nftables` таким образом:

```
sudo systemctl enable nftables
sudo systemctl start nftables
```

Чтобы проверить состояние `nftables`, запустите такую команду:

```
[packt@saturn2 ~]$ sudo systemctl status nftables
● nftables.service - Netfilter Tables
   Loaded: loaded (/usr/lib/systemd/system/nftables.service; enabled; preset:
disabled)
   Drop-In: /usr/lib/systemd/system/service.d
            └─10-timeout-abort.conf, 50-keep-warm.conf
   Active: active (exited) since Fri 2025-03-14 00:16:53 CET; 2h 32min ago
  Invocation: d7859e0d764a4286bacdd74f65719c1f
     Docs: man:nft(8)
   Process: 11838 ExecStart=/sbin/nft -f /etc/sysconfig/nftables.conf
(code=exited, status=0/SUCCESS)
    Main PID: 11838 (code=exited, status=0/SUCCESS)
   Mem peak: 3.1M
      CPU: 10ms
```

```
Mar 14 00:16:53 localhost.localdomain systemd[1]: Starting nftables.service -
Netfilter Tables...
```

```
Mar 14 00:16:53 localhost.localdomain systemd[1]: Finished nftables.service -
Netfilter Tables.
```


Если состояние службы — `active`, значит, мы готовы приступить к упражнениям по настройке `nftables`.

nftables: практические упражнения

Служба `nftables` загружает свою конфигурацию из файла `/etc/sysconfig/nftables.conf`. Его содержимое можно просмотреть с помощью знакомой вам команды `cat`:

```
[packt@saturn2 ~]$ sudo cat /etc/sysconfig/nftables.conf
# Uncomment the include statement here to load the default config sample
# in /etc/nftables for nftables service.

#include "/etc/nftables/main.nft"

# To customize, either edit the samples in /etc/nftables, append further
# commands to the end of this file or overwrite it after first service
# start by calling: 'nft list ruleset >/etc/sysconfig/nftables.conf'.
```

По умолчанию в конфигурационном файле нет активных записей, а есть только несколько комментариев о том, как изменить настройки `nftables`. Это можно сделать несколькими способами:

- напрямую отредактировать файл `nftables.conf`;
- отредактировать типовой конфигурационный файл `/etc/nftables/main.nft`, а затем раскомментировать соответствующую строку в файле `nftables.conf`;
- редактировать правила с помощью утилиты командной строки `nft`, а затем сохранить текущую конфигурацию в файле `nftables.conf`.

В любом случае после того, как вы измените конфигурацию, вам понадобится перезапустить службу `nftables`, чтобы применить обновленные настройки. В этом разделе мы воспользуемся утилитой `nft`. Опытные пользователи обычно пишут сценарии конфигурации `nftables`, но лучше сначала изучить основные шаги.

Следующая команда отображает все правила в текущей конфигурации:

```
sudo nft list ruleset
```

Возможно, в вашей системе уже настроены некоторые правила по умолчанию. Прежде чем переходить к следующему шагу, имеет смысл сохранить имеющуюся конфигурацию (например, сделать резервные копии файлов `/etc/sysconfig/nftables.conf` и `/etc/nftables/main.nft`).

Следующая команда удалит все ранее существовавшие правила:

```
sudo nft flush ruleset
```

Итак, теперь у нас пустая конфигурация. Давайте настроим простой межсетевой экран, который будет принимать трафик SSH, HTTP и HTTPS и блокировать все остальное.

Правила для обработки трафика SSH, HTTP и HTTPS

Сначала нужно создать таблицу, а в ней — цепочку. Следующая команда создаст таблицу `packt_table`:

```
sudo nft add table inet packt_table
```

Затем внутри этой таблицы можно создать цепочку `packt_chain`:

```
sudo nft add chain inet packt_table packt_chain { type filter hook input
priority 0 \; }
```

Теперь давайте добавим правила в эту цепочку. Чтобы разрешить доступ по SSH, HTTP и HTTPS, запустите такую команду:

```
sudo nft add rule inet packt_table packt_chain tcp dport {ssh, http, https}
accept
```

Стоит также включить протокол ICMP, который нужен, в частности, для команды `ping`:

```
sudo nft add rule inet packt_table packt_chain ip protocol icmp accept
```

Наконец, зададим правило, которое отклоняет весь остальной трафик:

```
sudo nft add rule inet packt_table packt_chain reject with icmp type port-
unreachable
```

Теперь давайте посмотрим на нашу новую конфигурацию:

```
[packt@saturn2 ~]$ sudo nft list ruleset
...
table inet packt_table {
    chain packt_chain {
        type filter hook input priority filter; policy accept;
        tcp dport { 22, 80, 443 } accept
            # Разрешать трафик TCP на портах назначения 22, 80 и 43
        ip protocol icmp accept
            # Разрешать запросы ping
        reject with icmp port-unreachable
            # Отклонять все остальное
    }
}
```

Сохраните текущую конфигурацию в новом файле `/etc/nftables/packt.nft`:

```
sudo nft list ruleset | sudo tee /etc/nftables/packt.nft
```

После этого вставьте ссылку на этот файл в основной конфигурационный файл `/etc/sysconfig/nftables.conf`. Для этого добавьте в конец файла `nftables.conf` такую строку:

```
include "/etc/nftables/packt.nft"
```

Чтобы это сделать, можно отредактировать конфигурационный файл в текстовом редакторе (например, Vim или nano), а можно запустить такую команду:

```
echo 'include "/etc/nftables/packt.nft"' | sudo tee -a /etc/sysconfig/nftables.conf
```

Наконец, давайте перезапустим службу `nftables`, чтобы применить новую конфигурацию:

```
sudo systemctl restart nftables
```

После этого упражнения вам будет легче самостоятельно написать сценарий для настройки `nftables`, используя вывод команды `nft list ruleset`. Собственно говоря, мы только что сделали это с помощью конфигурационного файла `/etc/nftables/packt.nft`.

На этом мы заканчиваем рассматривать системы фильтрации пакетов и соответствующие утилиты командной строки. Эти инструменты дают опытным пользователям полноценный контроль над каждым функциональным аспектом сетевых цепочек и правил. Однако для многих повседневных задач хватает более простых инструментов, которые тоже позволяют управлять межсетевым экраном. Далее мы рассмотрим две системы управления межсетевыми экранами, которые встроены в Linux и предоставляют более упрощенный и удобный интерфейс командной строки.

Системы управления межсетевыми экранами

Системы управления межсетевыми экранами — это утилиты командной строки, которые позволяют настраивать правила безопасности межсетевого экрана с помощью относительно простого интерфейса. Инструменты такого рода весьма востребованы среди системных администраторов Linux и обычно требуют привилегий суперпользователя.

В следующих разделах мы расскажем о двух самых распространенных системах, которые широко используются в современных дистрибутивах Linux:

- `firewalld` (RHEL/Fedora);
- `ufw` (Ubuntu/Debian).

Системы управления межсетевыми экранами в той или иной степени выполняют те же функции, что и более сложные инструменты сетевой безопасности (такие, как Netfilter, `iptables` и `nftables`), но абстрагируют многие сложности и предлагают простой и удобный интерфейс. Существенное преимущество таких систем состоит в том, что они позволяют модифицировать настройки безопасности, не перезапуская сетевые демоны.

Начнем с `firewalld` — стандартной системы управления межсетевыми экранами в RHEL/Fedora.

Как работать с *firewalld*

firewalld — это система управления межсетевыми экранами по умолчанию в различных дистрибутивах Linux, в том числе:

- RHEL (7 и новее);
- openSUSE (15 и новее);
- Fedora (18 и новее).

Если в вашей системе семейства RHEL нет пакета *firewalld*, его можно установить с помощью такой команды:

```
sudo yum install -y firewalld
```

Если ранее вы отключали *firewalld* (например, чтобы выполнять упражнения с *iptables* из предыдущих разделов), его снова следует включить и убедиться, что он работает. Это можно сделать так:

```
[packt@saturn2 ~]$ sudo systemctl unmask firewalld
Removed '/etc/systemd/system/firewalld.service'.
```

```
[packt@saturn2 ~]$ sudo systemctl enable firewalld
Created symlink '/etc/systemd/system/dbus-org.fedoraproject.FirewallD1.service'
→ '/usr/lib/systemd/system/firewalld.service'.
Created symlink '/etc/systemd/system/multi-user.target.wants/firewalld.service'
→ '/usr/lib/systemd/system/firewalld.service'.
```

```
[packt@saturn2 ~]$ sudo systemctl start firewalld
```

```
[packt@saturn2 ~]$ sudo systemctl status firewalld
● firewalld.service - firewalld - dynamic firewall daemon
   Loaded: loaded (/usr/lib/systemd/system/firewalld.service; enabled; preset:
enabled)
   Drop-In: /usr/lib/systemd/system/service.d
            └─10-timeout-abort.conf, 50-keep-warm.conf
   Active: active (running) since Fri 2025-03-14 17:20:04 CET; 3s ago
 Invocation: c8d5951b547e41c7bb3e624de1cf9b9b
    Docs: man:firewalld(1)
   Main PID: 12135 (firewalld)
     Tasks: 2 (limit: 2211)
    Memory: 28.8M (peak: 29M)
       CPU: 195ms
    CGroup: /system.slice/firewalld.service
            └─12135 /usr/bin/python3 -sP /usr/bin/firewalld --nofork --nopic
```

```
Mar 14 17:20:04 localhost.localdomain systemd[1]: Starting firewalld.service
- firewalld - dynamic firewall daemon...
Mar 14 17:20:04 localhost.localdomain systemd[1]: Started firewalld.service -
firewalld - dynamic firewall daemon.
```

В *firewalld* входит набор утилит командной строки для различных задач:

firewall-cmd	Основной инструмент командной строки
firewall-offline-cmd	Позволяет настраивать firewalld , когда он отключен (например, во время установки системы)
firewall-config	Графический интерфейс для настройки firewalld
firewall-applet	Приложение для панели уведомлений, предоставляющее важную информацию о firewalld (например, сведения о состояниях службы и соединений)

В этом разделе мы рассмотрим несколько практических примеров использования **firewall-cmd**. Чтобы получить дополнительную информацию о других утилитах, можно обратиться к соответствующим справочным страницам (например, `man firewall-config`).

firewalld (а значит, и **firewalld-cmd**) опирается на несколько ключевых понятий, которые относятся к мониторингу и контролю сетевых пакетов: *зоны*, *правила* и *цели*.

Зоны **firewalld**

Зоны — это верхнеуровневые структурные единицы конфигурации **firewalld**. Сетевой пакет, который отслеживает **firewalld**, считается принадлежащим зоне, если он соответствует сетевому интерфейсу или IP-адресу (возможно, с маской сети), который связан с зоной. Следующая команда перечисляет имена зон, которые определены по умолчанию:

```
sudo firewall-cmd --get-zones
```

Чтобы получить подробные сведения обо всех существующих зонах, можно запустить такую команду:

```
[packt@saturn2 ~]$ sudo firewall-cmd --list-all-zones
FedoraServer (default, active)
  target: default
  ingress-priority: 0
  egress-priority: 0
  icmp-block-inversion: no
  interfaces: ens160
  sources:
  services: cockpit dhcpv6-client ssh
  ports:
  protocols:
  forward: yes
  masquerade: no
  forward-ports:
  source-ports:
  icmp-blocks:
  rich rules:
... и т. д. ...
```

Этот пример демонстрирует активную зону по умолчанию в серверной версии Fedora 37 и перечисляет атрибуты зоны; некоторые из них мы рассмотрим далее. Зоны, для которых задан интерфейс и источник, называются **активными**. Следующая команда выводит все активные зоны:

```
[packt@saturn2 ~]$ sudo firewall-cmd --get-active-zones
FedoraServer (default)
  interfaces: ens160
```

Интерфейсы — это сетевые адаптеры, которые подключены к локальному узлу. Активные интерфейсы привязываются либо к зоне по умолчанию, либо к зоне, которую определил пользователь. Один и тот же интерфейс нельзя привязать к нескольким зонам.

Источники — это входящие IP-адреса или диапазоны адресов, которые тоже можно привязывать к зонам. Один и тот же источник (или несколько перекрывающихся диапазонов IP-адресов) нельзя привязать более чем к одной зоне. Это позволяет предотвратить неопределенное поведение, когда неясно, какое правило приоритетнее для той или иной зоны.

По умолчанию `firewalld` привязывает все сетевые интерфейсы к зоне `public`, но не ассоциирует с ней никаких источников. Кроме того, по умолчанию `public` является единственной активной зоной и, следовательно, зоной по умолчанию. Следующая команда отображает зону по умолчанию в вашей системе:

```
sudo firewall-cmd --get-default-zone
```

Указывать источник для зоны необязательно. Если он не указан, то каждому пакету данных будет соответствовать зона с тем сетевым интерфейсом, через который он поступил, но не обязательно будет соответствовать зона с конкретным источником. Этот принцип будет важен для того, в каком порядке пакет сопоставляется с правилами. Мы обсудим эту тему далее в разделе «Приоритет правил», но сначала давайте познакомимся с правилами `firewalld`.

Правила `firewalld`

Правила, которые определены в конфигурации `firewalld`, представляют собой настройки конфигурации для пакетов данных, которые связаны с определенной зоной. Обычно правило определяет (на основе тех или иных заранее заданных критериев), будет ли пакет принят или отклонен.

Например, чтобы заблокировать запросы `ping` (по протоколу ICMP) для зоны `FedoraServer`, можно добавить такой атрибут `add-rich-rule`:

```
[packt@saturn2 ~]$ sudo firewall-cmd --zone=FedoraServer --add-rich-rule='rule
protocol value="icmp" reject'
success
```

После этого можно запросить информацию о зоне, чтобы убедиться, что правило добавлено:

```
[packt@saturn2 ~]$ sudo firewall-cmd --info-zone=FedoraServer
FedoraServer (default, active)
  target: default
  ... И Т. Д. ...
  rich rules:
    rule protocol value="icmp" reject
```

С этого момента наш узел перестанет отвечать на запросы ping (ICMP). Правило, которое мы только что добавили, можно снова отменить с помощью такой команды:

```
sudo firewall-cmd --zone=FedoraServer --remove-rich-rule='rule protocol
value="icmp" reject'
```

Вместно этого можно было специально включить доступ по протоколу ICMP с помощью такой команды:

```
sudo firewall-cmd --zone=FedoraServer --add-rich-rule='rule protocol
value="icmp" accept'
```

Обратите внимание, что изменения, которые мы вносим с помощью `firewall-cmd` без ключа `--permanent`, носят временный характер и не сохраняются, если перезапустить `firewalld` (в том числе вместе с перезагрузкой всей системы).

Если для зоны не определены правила и/или если пакет не соответствует ни одному правилу, то `firewalld` управляет поведением пакета на основе цели зоны. Давайте поговорим о том, что такое цели.

Цели `firewalld`

Когда пакет соответствует определенной зоне, `firewalld` обрабатывает его в соответствии с правилами этой зоны. Но если правила не определены или если пакет не соответствует ни одному из них, то его поведение в конечном итоге определяется **целью**, которая связана с зоной. Вот возможные цели:

ACCEPT	Принять пакет
REJECT	Отклонить пакет и ответить отказом
DROP	Отбросить пакет без ответа
default	По умолчанию; подобно REJECT, но с некоторыми особенностями ¹

Зоны, правила и цели — это ключевые компоненты конфигурации, на основе которых `firewalld` анализирует и обрабатывает пакеты данных. Пакеты сопоставляются с зонами, а затем обрабатываются с помощью правил или целей.

¹ Официальная справочная страница (`man firewallld.zone`) не вполне ясно документирует поведение по умолчанию, но в обсуждении проекта `firewalld` на GitHub предлагается дополнение к справочной странице, в котором исчерпывающе описано это поведение: github.com/firewalld/firewalld/issues/590#issuecomment-605200548. — Примеч. ред.

Поскольку зоны могут быть связаны и с сетевыми интерфейсами, и с IP-адресами или диапазонами адресов источников, `firewalld` сопоставляет пакеты с критериями в определенном порядке, который мы рассмотрим далее.

Приоритет правил

Сначала определимся с терминологией. Если с зоной связан интерфейс, мы будем называть ее **интерфейсной зоной**, а если с ней связаны адреса источников, — **адресной зоной**. Если с зоной связан и интерфейс, и источники, она считается интерфейсной и адресной одновременно.

`firewalld` обрабатывает пакеты данных в таком порядке:

1. Сначала он ищет соответствующую адресную зону. Таких зон может быть не более одной (потому что каждый источник привязывается только к одной зоне). Если зона найдена, пакет обрабатывается в соответствии с правилами или целью, которые определены для этой зоны, иначе обработка переходит к следующему шагу.
2. `firewalld` ищет соответствующую интерфейсную зону. Всегда существует ровно одна такая зона, и пакет обрабатывается в соответствии с ее правилами или целью.

В итоге по умолчанию `firewalld` принимает пакеты ICMP и отклоняет все остальные. Основной вывод здесь в том, что адресные зоны имеют приоритет над интерфейсными. В типичной многозоновой конфигурации `firewalld` определяются такие зоны:

- **Привилегированная адресная зона.** Расширенный доступ к системе с избранных IP-адресов.
- **Ограниченная интерфейсная зона.** Ограниченный доступ для всех остальных.

Давайте рассмотрим еще несколько полезных примеров того, что можно сделать с помощью `firewall-cmd`:

- Отобразить службы, которые включены в межсетевом экране:

```
sudo firewall-cmd --list-services
```
- Включить для определенной зоны доступ по протоколу HTTPS (порт 443):

```
sudo firewall-cmd --zone=FedoraServer --add-service=https
```
- Добавить определенную службу или порт (например, 8443):

```
sudo firewall-cmd --zone=FedoraServer --add-port=8443/tcp
```
- Перечислить порты, которые открыты в межсетевом экране:

```
sudo firewall-cmd --list-ports
```
- Вернуться к ранее сохраненной (постоянной) конфигурации `firewalld`:

```
sudo firewall-cmd --reload
```


Чтобы получить дополнительную информацию о `firewalld`, обратитесь к соответствующей справочной странице (`man firewalld`) или к официальному сайту `firewalld.org`.

Как работать с `ufw`

`ufw`¹ — это система управления межсетевыми экранами по умолчанию в Ubuntu. Она служит надстройкой над подсистемами `iptables/nftables` и предоставляет удобный интерфейс командной строки для того, чтобы работать с межсетевыми экранами.

Давайте рассмотрим несколько примеров использования `ufw`. Обратите внимание, что для этой утилиты нужны права суперпользователя. Следующая команда сообщает о состоянии `ufw`:

```
sudo ufw status
```

По умолчанию служба `ufw` отключена (`Status: inactive`). Ее можно включить с помощью такой команды:

```
sudo ufw enable
```

Всегда будьте осторожны, когда включаете межсетевой экран или выполняете другие операции, которые могут повлиять на ваш доступ к системе. Если включить `ufw`, по умолчанию она блокирует весь входящий трафик, кроме запросов `ping` (ICMP). Если вы вошли в систему по SSH, то при попытке включить `ufw` может появиться предупреждение о том, что соединение SSH может быть разорвано:

```
Command may disrupt existing ssh connections. Proceed with operation (y|n)?
```

В этом случае, чтобы перестраховаться, можно прервать предыдущую операцию, нажав `n` («Нет»), а затем включить доступ по SSH в межсетевом экране:

```
packt@neptune:~$ sudo ufw allow ssh
Rules updated2
Rules updated (v6)
```

Если доступ к SSH уже был включен, эта команда ничего не испортит, а только выведет соответствующее уведомление.

После этого можно спокойно включать `ufw`, не опасаясь, что ваше текущее соединение (или другие соединения SSH) будет разорвано. В этом случае в ответ на предупреждение можно смело нажимать `y` («Да»).

¹ Название `ufw` — сокр. от Uncomplicated Firewall (межсетевой экран без осложнений). — *Примеч. пер.*

² Правила обновлены. — *Примеч. пер.*

Чтобы просмотреть подробные сведения о состоянии межсетевого экрана, можно запустить такую команду:

```
packt@neptune:~$ sudo ufw status verbose
Status: active
Logging: on (low)
Default: deny (incoming), allow (outgoing), disabled (routed)
New profiles: skip
```

To	Action	From
--	-----	----
22/tcp	ALLOW IN	Anywhere
22/tcp (v6)	ALLOW IN	Anywhere (v6)

Всегда рекомендуется проверять настройки межсетевого экрана, чтобы исключить возможность непреднамеренного доступа к системе.

Чтобы перечислить текущие профили безопасности приложений, запустите такую команду:

```
sudo ufw app list
```

В нашем случае доступен только OpenSSH, который был активирован, когда мы разрешили соединения по SSH ранее в этом разделе.

Давайте добавим другие протоколы — например, HTTP (порт 80) и HTTPS (порт 443), которые нужны для веб-серверов Apache и nginx. Чтобы включить соответствующие службы, можно указать номера портов (80, 443), имена служб (http, https) или напрямую идентификаторы веб-серверов (Apache Full, Nginx Full), например:

```
packt@neptune:~$ sudo ufw allow http; sudo ufw allow https
Rule added
Rule added (v6)
Rule added
Rule added (v6)
```

После этого команда `ufw status` отобразит изменения:

```
packt@neptune:~$ sudo ufw status
Status: active
```

To	Action	From
--	-----	----
22/tcp	ALLOW	Anywhere
80/tcp	ALLOW	Anywhere
443	ALLOW	Anywhere
22/tcp (v6)	ALLOW	Anywhere (v6)
80/tcp (v6)	ALLOW	Anywhere (v6)
443 (v6)	ALLOW	Anywhere (v6)

Чтобы удалить доступ к определенной службе, запустите команду `ufw deny`. Например, так можно удалить доступ к HTTP:

```
sudo ufw deny http
```

Эта команда выведет стандартное сообщение о том, что было добавлено новое правило, а `ufw status` покажет, что доступ к порту `80/tcp` запрещен (приведем только те строки, которые изменились):

```
packt@neptune:~$ sudo ufw status
To                Action            From
--                -
80/tcp            DENY              Anywhere
80/tcp (v6)       DENY              Anywhere (v6)
```

Такой вывод ничего не сообщает о том, в каком порядке применяются правила межсетевого экрана. Чтобы увидеть этот порядок, можно добавить аргумент `numbered`:

```
rostislav@ubuntu:~$ sudo ufw status numbered
Status: active

      To                Action            From
      --                -
[ 1] 22/tcp            ALLOW IN          Anywhere
[ 2] 80/tcp            DENY IN          Anywhere
[ 3] 443               ALLOW IN          Anywhere
[ 4] 22/tcp (v6)       ALLOW IN          Anywhere (v6)
[ 5] 80/tcp (v6)       DENY IN          Anywhere (v6)
[ 6] 443 (v6)         ALLOW IN          Anywhere (v6)
```

В идеале, чем более специфично ограничительное правило, тем ближе к началу списка его нужно размещать. По мере того как вы добавляете или изменяете правила, вам может понадобиться удалить старые правила или расставить их в другом порядке.

Чтобы добавить определенное правило в заданную позицию, можно использовать аргумент `insert`. Например, следующая команда запрещает подключение по TELNET и помещает соответствующее правило во вторую позицию:

```
sudo ufw insert 2 deny telnet
```

Рассмотрим еще несколько упражнений с `ufw`. Следующая команда разрешает доступ по SSH (порт 22) к любому локальному адресу (`any`) из определенного диапазона адресов источника (`192.168.0.0/24`):

```
sudo ufw allow from 192.168.0.0/24 to any port 22
```

Следующая команда включает ведение журнала `ufw`:

```
sudo ufw logging on
```

Соответствующие записи журнала обычно можно найти в файле `/var/log/syslog`:

```
grep -i ufw /var/log/syslog
```

Чтобы отключить ведение журнала `ufw`, запустите такую команду:

```
sudo ufw logging off
```

Наконец, следующая команда возвращает `ufw` к настройкам по умолчанию, то есть удаляет все правила и отключает службу `ufw`:

```
sudo ufw reset
```

Чтобы получить дополнительную информацию об `ufw`, можно обратиться к вики-справочнику по адресу help.ubuntu.com/community/UFW или соответствующей справочной странице (`man ufw`).

Часто системным администраторам бывает удобнее управлять межсетевыми экранами с помощью таких инструментов, как `ufw` и `firewalld`, чем с помощью более низкоуровневых утилит фильтрации пакетов (например, `iptables` и `nftables`). Выбор того или иного инструмента зависит не только от платформы, но и от того, насколько эффективно он позволяет писать сценарии и автоматизировать задачи. Некоторые продвинутые пользователи предпочитают настраивать правила межсетевого экрана утилитой `nft`, которая обеспечивает детальный контроль, характерный для `nftables`. Другие пользователи склоняются к `iptables`, особенно на старых платформах. В конце концов, многое зависит от личных предпочтений, потому что все эти инструменты так или иначе позволяют добиться одних и тех же результатов в области управления межсетевыми экранами.

Итоги

Смеем надеяться, что эта довольно обширная глава вас не слишком утомила. Главное, что стоило из нее извлечь, — представление о фреймворках (модулях) безопасности. Если заходит речь о межсетевых экранах, нужно обратить внимание на внутриядерный фреймворк фильтрации пакетов Netfilter и связанные с ним утилиты пользовательского пространства `iptables` и `nftables`. Для управления доступом есть такие модули безопасности, как SELinux и AppArmor; мы рассмотрели некоторые плюсы и минусы каждого из них. Возможно, от выбора между этими службами в конечном итоге будет зависеть, на каком дистрибутиве Linux вы остановитесь. Например, если вы предпочтете AppArmor, то круг доступных дистрибутивов Linux сузится до семейства, к которому принадлежат Ubuntu, Debian и openSUSE. В свою очередь, от дистрибутива будет зависеть, какие решения для управления межсетевыми экранами будут вам доступны, и т. д.

Если вы уверенно освоите модули безопасности и системы управления межсетевыми экранами, то сможете обеспечивать защиту системы с минимальными усилиями. Как и любую другую типичную задачу системного администрирования Linux, эту работу можно выполнить многими разными способами. Мы на-

деемся, что сведения и инструменты, о которых шла речь в этой главе, помогут вам принимать взвешенные решения о том, как обеспечивать безопасность Linux.

В следующей главе вы узнаете, что такое **аварийное восстановление**, а также как диагностировать и устранять неполадки.

Упражнения

Вот небольшой тест на то, как вы усвоили основные понятия, которые рассматривались в этой главе:

1. Назовите несколько типов систем управления доступом, которые используются в Linux.

Подсказка: DAC, ACL, MAC, RBAC, MLS, MCS.

2. Перечислите поля контекста безопасности SELinux.

Подсказка: пользователь, роль, тип, уровень.

3. Что такое домен в SELinux?

Подсказка: это тип, к которому относится процесс.

4. В чем существенные различия между SELinux и AppArmor с точки зрения того, как применяются политики безопасности?

Подсказка: политики в SELinux основаны на метках файлов, а в AppArmor — на путях в файловой системе.

5. Как переключить профиль приложения AppArmor между принудительным и уведомительным режимами?

Подсказка: используйте команды `aa-enforce` и `aa-complain`.

6. Сколько цепочек в сетевом стеке ядра Linux вы можете назвать?

Подсказка: вам может помочь таблица на с. 372–373.

7. Какая система управления межсетевыми экранами используется по умолчанию в RHEL/Fedora? А в Ubuntu?

Подсказка: `firewalld` (Fedora) и `ufw` (Ubuntu).

Дополнительная литература

Чтобы подробнее познакомиться с темами, которые рассматривались в этой главе, обратитесь к следующим книгам издательства Packt:

- Дональд А. Тиволт (Donald A. Tevault). «Mastering Linux Security and Hardening, Second Edition»¹.
- Таджиндер Калси (Tajinder Kalsi). «Practical Linux Security Cookbook, Second Edition».

¹ Тиволт Д. «Защита и укрепление Linux».

10

Аварийное восстановление, диагностика и устранение неполадок

В этой главе вы узнаете, как выполнять резервное копирование и восстанавливать систему после аварийных сбоев, а также как **диагностировать** и **устранять** распространенные неполадки. Этими навыками должен обладать каждый системный администратор Linux, чтобы не растеряться в наихудших сценариях, таких как отключение электричества, кража или отказ оборудования. Большая часть компьютерной инфраструктуры во всем мире работает на Linux, и нам с вами стоит быть готовыми к самым разным техническим сюрпризам.

В этой главе мы рассмотрим такие темы:

- Как планировать аварийное восстановление.
- Как создавать резервные копии и восстанавливать систему.
- Как устранять неполадки с помощью распространенных диагностических инструментов Linux.

Технические требования

Для этой главы не требуется никаких особых технических условий. Достаточно, чтобы у вас была установлена действующая система Linux, а для некоторых примеров пригодятся две разные системы в одной и той же локальной сети. Для упражнений этой главы одинаково хорошо подходят Ubuntu и Fedora, хотя мы будем демонстрировать все примеры в серверной и настольной версиях Ubuntu 22.04.2 LTS.

Как планировать аварийное восстановление

Каждому предприятию и даже отдельному человеку важно уметь управлять рисками. В этом отношении на всех, кто занимается системным администрированием, лежит особенно большая ответственность. Как правило, в бизнесе разрабатываются и внедряются эффективные **стратегии управления рисками**. В сфере IT приходится учитывать различные виды рисков — начиная от стихийных бедствий,

которые наносят физический ущерб центрам обработки данных и инфраструктуре предприятий, и заканчивая угрозами кибербезопасности. За последнее десятилетие влияние ИТ в компаниях выросло до небывалого уровня. Сегодня нет ни одного вида деятельности, который в той или иной форме не опирается на компьютерные технологии: они задействуются в малом бизнесе, крупных корпорациях, государственных учреждениях, здравоохранении, образовании и где угодно еще. Каждый вид деятельности по-своему уникален, а значит, в каждой области стоит учитывать свои особые факторы риска. К сожалению, в сфере информационной безопасности управление рисками в значительной степени выродилось в унифицированную формальную практику с контрольными списками, которые должно внедрять руководство отдела ИТ. Давайте начнем с краткого введения в управление рисками, а затем поговорим о том, как их оценивать.

Краткое введение в управление рисками

Что такое «управлять рисками»? В двух словах это означает принимать превентивные меры для того, чтобы смягчить возможные угрозы, которые могут нарушить непрерывную работу бизнеса. Управление рисками крайне важно для любого отдела ИТ.

Системы управления рисками в сфере компьютерных технологий появились в США благодаря закону **о модернизации федеральных информационных систем (FISMA)**, который начал действовать в 2002 году. Именно тогда **Национальный институт стандартов и технологий США (NIST)** начал создавать новые стандарты и методы оценки кибербезопасности для всех государственных учреждений. Поэтому сертификация и соответствие требованиям безопасности крайне важны для каждого поставщика дистрибутива Linux, который хочет быть достойным конкурентом в корпоративном и правительственном сегменте. В других странах тоже существуют государственные организации, которые курируют сертификацию в области кибербезопасности. Неудивительно, что все основные дистрибутивы Linux от Red Hat, SUSE и Canonical имеют сертификаты NIST, **NCSC** (Великобритания) или **ФСТЭК** (Россия).

Стандарт NIST SP 800-37 Rev. 2 (csrc.nist.gov/pubs/sp/800/37/r2/final) предлагает семиступенчатый процесс управления рисками, начиная с подготовительных мероприятий и заканчивая регулярным мониторингом систем предприятия. Здесь мы не будем подробно рассматривать эти этапы, а ограничимся ссылкой на официальную документацию NIST в конце главы. В двух словах, система управления рисками охватывает такие ключевые направления:

- **Инвентаризация.** Подробная опись всех имеющихся локальных систем и программных решений.
- **Категорирование объектов системы.** Распределение ресурсов и других объектов по классам опасности с точки зрения доступности, целостности и конфиденциальности.

- **Элементы управления безопасностью.** Административные, технические и физические меры безопасности; сборник элементов управления NIST можно найти в стандарте SP 800-53 Rev. 5 (csrc.nist.gov/pubs/sp/800/53/r5/upd1/final).
- **Оценка рисков.** Мероприятия, направленные на то, чтобы обнаружить источники угроз, выявить уязвимости, оценить потенциальный ущерб, а затем поддерживать эту информацию в актуальном состоянии.
- **План обеспечения безопасности системы.** Правила о том, как применять каждый элемент управления безопасностью и какие действия предпринимать в будущем, включая их реализацию и оценку эффективности.
- **Сертификация, аккредитация, оценка и авторизация.** Задача этого направления — анализировать результаты оценки безопасности, выделять проблемы безопасности и вырабатывать эффективные решения, которые подробно описываются в плане дальнейших действий.
- **План действий.** Инструмент, который позволяет отслеживать слабые места в системе безопасности и применять правильные процедуры реагирования.

В области информационных технологий бывает много разновидностей рисков, включая сбои оборудования, ошибки программного обеспечения, спам, вирусы, человеческий фактор и стихийные бедствия (пожары, наводнения, землетрясения, ураганы и т. д.). Существуют также риски криминального характера, такие как взломы, недобросовестные действия сотрудников, корпоративный шпионаж и все остальное, что обычно относится к киберпреступности. Чтобы снизить эти риски, можно внедрить стратегию управления рисками, взяв за основу пять таких этапов:

1. **Идентификация рисков.** Выявляются возможные угрозы и уязвимости, которые могут повлиять на текущие операции в системе.
2. **Анализ рисков.** На основе тщательных исследований вырабатывается представление о том, насколько велик риск и к каким последствиям он может привести.
3. **Оценивание рисков.** Определяется значимость риска и вырабатываются соответствующие меры реагирования.
4. **Реагирование на риски.** Внедряются **планы аварийного восстановления**, а также стратегии предотвращения и смягчения последствий.
5. **Мониторинг рисков.** Реализуется действенная стратегия мониторинга, которая позволяет убедиться, что все нужные сотрудники и подразделения знают, как реагировать на риск, и что у них есть инструменты и возможности для того, чтобы изолировать риск и поддерживать безопасность инфраструктуры компании.

В любом бизнесе чрезвычайно важно адекватно оценивать риски, и техническое руководство должно относиться к этой задаче со всей серьезностью. Теперь, когда мы рассмотрели некоторые понятия из области управления рисками, пришло время объяснить, что же это такое на самом деле.

Расчет рисков

Оценка рисков (или расчет рисков) заключается в том, что мы вырабатываем и численно обосновываем решения для защиты от возможных угроз и уязвимостей. Вот некоторые термины, которые стоит знать, чтобы рассчитывать риски¹:

- **Предполагаемые годовые потери (ALE)** — убытки, которые ожидаются в течение 1 года.
- **Предполагаемая единичная потеря (SLE)** — убытки, которые ожидаются от однократной реализации риска.
- **Годовая частота возникновения (ARO)** — предполагаемое количество случаев реализации риска в течение 1 года.
- **ALE** вычисляется так: $ALE = SLE \times ARO$. Это полезная формула, которая дает предполагаемые годовые потери в денежном выражении.
- **Средняя наработка между отказами (MTBF)** характеризует время между последовательными устранимыми отказами.
- **Средняя наработка на отказ (MTTF)** характеризует время, в течение которого система может работать, прежде чем произойдет неустраняемый отказ.
- **Среднее время восстановления (MTTR)** — это время, которое необходимо, чтобы восстановить поврежденную систему.
- **Целевой срок восстановления (RTO)** — максимальное плановое время, в течение которого система может простаивать.
- **Целевая точка восстановления (RPO)** — объем данных, который допускается потерять при аварии, выраженный в виде временного отрезка (например, «система допускает потерю данных за последние 1,5 часа»).

Эти понятия помогут оценивать риски, чтобы в случае необходимости провести хорошо документированную оценку. Когда необходимо реагировать на риски, соответствующие меры обычно выбирают из таких стратегий:

- **Проактивные действия:**
 - **Избегать рисков.** Выявлять возможные риски и находить способы их избежать.
 - **Снижать риски.** Принимать меры для того, чтобы уменьшить вероятность возникновения риска.
 - **Передать риски.** Передать возможные последствия риска внешнему субъекту.

¹ В оригинальном издании авторы иногда немного вольно обращаются с темами, терминами и понятиями из области информационной безопасности. Мы предлагаем читателям сверяться с официальными источниками: например, уточнять значения терминов в справочнике NIST (csrc.nist.gov/glossary). Перевод на русский язык приведен в соответствии с отечественными государственными стандартами (например, ГОСТ Р 27.102-2021 или ГОСТ Р 53647.4-2011). — *Примеч. ред.*

- **Сдерживать риск.** Применять специализированные системы и политики, которые должны помешать злоумышленникам использовать систему в своих целях.
- **Неактивные действия:**
 - **Принять риск.** Смириться с риском, если проактивные действия могут обойтись дороже, чем ущерб, который нанесет риск.

Описанные здесь стратегии можно применить к рискам, которые связаны с вычислениями на традиционных локальных компьютерах, но в последнее время мир уверенно переходит на облачные вычисления. Как же применить эти стратегии реагирования к облачным технологиям? В этом случае вы используете инфраструктуру третьей стороны, которая работает с вашими собственными данными. Хотя мы начнем обсуждать облачные решения Linux в главе 14 «Краткое введение в облачные вычисления», некоторые понятия мы представим уже сейчас. Как мы уже говорили, облачные платформы переносят инфраструктурные операции из вашей локальной среды к специализированному провайдеру, такому как Amazon, Microsoft или Google. В целом это можно рассматривать как аутсорсинг. Это значит, что определенные риски, которые представляли угрозу для локальных служб, теперь передаются третьим лицам.

Во всех технических сообществах сейчас на слуху три основные парадигмы облачных вычислений:

- **Программное обеспечение как услуга (SaaS).** Программное решение для компаний, которые хотят сократить расходы на ИТ и пользоваться программным обеспечением по подписке. Примеры решений SaaS — **Slack, Microsoft 365, Google Apps** и **Dropbox**.
- **Платформа как услуга (PaaS).** Поставщик доставляет свои прикладные программы клиентам с помощью чужой инфраструктуры, среды выполнения и зависимостей. Это может быть общедоступное облако, частное облако или гибридное решение. Примеры PaaS — **Microsoft Azure, AWS Lambda, Google App Engine, SAP Cloud Platform, Heroku** и **Red Hat OpenShift**.
- **Инфраструктура как услуга (IaaS).** Службы, которые доступны через интернет и предоставляют высокоуровневые **API**, например **OpenStack**.

Подробнее обо всех этих платформах мы расскажем в главе 14 «Краткое введение в облачные вычисления», а для текущей главы вполне хватит того, что мы уже рассмотрели. Основные риски облачных вычислений связаны с интеграцией данных и совместимостью. Это те немногие риски, которыми вам придется управлять самостоятельно, а большинство других рисков вас уже не касаются, потому что они передаются третьей стороне, которая управляет инфраструктурой.

Оценивать риски можно по-разному в зависимости от того, как устроена инфраструктура ИТ в вашей компании. Если вы используете локальные мощности и самостоятельно управляете всеми компонентами, оценивать риски может оказаться непростой задачей. Но если вы задействуете IaaS, PaaS и SaaS, оце-

нивать риски становится проще, потому что большая часть ответственности передается внешним подрядчикам.

Любому, кто заботится о безопасности своей сети и систем, а тем более руководителям отделов IT, стоит серьезно относиться к оценке рисков. Чем эффективнее удастся оценить риски, тем лучше будет план аварийного восстановления и соответствующая стратегия.

Как разрабатывать план аварийного восстановления

В плане аварийного восстановления (DRP) фиксируются шаги, которые необходимо предпринять, если произойдет аварийная ситуация. Чаще всего DRP входит в **план обеспечения непрерывности бизнеса**, который описывает, как компания должна продолжать работу на основе функционирующей инфраструктуры.

Всякий DRP должен начинаться с подробной **инвентаризации оборудования**, за которой следует **инвентаризация программного обеспечения** и отдельная **инвентаризация данных**. При этом крайне важно придерживаться надежной стратегии резервного копирования, которая охватывает всю рабочую информацию.

В том, что касается аппаратного обеспечения, необходимо наладить четкую политику по стандартизации оборудования, которая обеспечит легкую замену неисправного оборудования. Такая политика позволяет оптимизировать аппаратные ресурсы. Для стандартизированного оборудования характерна хорошая поддержка драйверов, а это очень важно в мире Linux. С другой стороны, в условиях стандартизации сотрудники лишены возможности использовать в работе собственные устройства и вынуждены обходиться только тем оборудованием, которое предоставляет работодатель. На стандартизированном оборудовании обычно также используется стандартизированное программное обеспечение, которое устанавливает и налаживает отдел IT, а обычные пользователи не могут настраивать ПО по своему усмотрению.

На отделе IT лежит огромная ответственность, и он играет ключевую роль в разработке **стратегий восстановления вычислительной инфраструктуры** в рамках DRP. Базовые допуски на время простоя и на потерю данных должны определяться на основе минимально допустимых RPO и RTO.

Для хорошего DRP не менее важно распределить **роли**, чтобы было понятно, кто за что отвечает. Благодаря этому компания сможет быстрее реагировать на нештатные ситуации и каждый сотрудник будет знать, что ему делать, если случится авария. В свою очередь, для этого важна хорошая **стратегия коммуникации**. Если для каждого уровня организационной пирамиды налажены четкие процедуры, это обеспечит бесперебойную коммуникацию, централизованные решения и план преемственности персонала.

Чтобы убедиться, что DRP эффективен, его необходимо тщательно тестировать не реже двух раз в год. Незапланированные простои и сбои могут повредить бизнесу как в локальной, так и в облачной среде. Важно всегда быть готовым

к наихудшим сценариям. В следующих разделах мы продемонстрируем некоторые эффективные инструменты и приемы для резервного копирования и восстановления систем Linux.

Резервное копирование и восстановление системы

Авария может произойти в любой момент, и ни одна система не застрахована от сбоев. Поэтому резервное копирование играет важнейшую роль, и его нужно выполнять регулярно. Всегда лучше потратить усилия на профилактику, чем потом восстанавливать утраченные данные и учиться на собственном печальном опыте.

Резервное копирование и восстановление должно опираться на хорошо продуманную стратегию и учитывать факторы RTO и RPO. Показатель RTO отвечает за то, насколько быстро будут восстановлены данные и как это повлияет на бизнес-операции, а RPO — за то, какой объем данных вы можете позволить себе потерять.

Существуют различные типы и методы резервного копирования. Вот некоторые примеры:

Типы резервного копирования		Что копируется	
Полное		Все файлы в целевом расположении	
Инкрементальное		Все файлы, которые изменились после предыдущего резервного копирования	
Дифференциальное		Все файлы, которые изменились после предыдущего полного резервного копирования	
Методы резервного копирования			
Вручную		Иницируется пользователем, не обязательно по жесткому графику	
Автоматически	Локально	Запускается автоматически по расписанию, надежнее ручного метода	Обработывает локальные накопители
	Удаленно		Обработывает также внешние накопители, подключенные по сети

Рис. 10.1. Методы и типы резервного копирования

Создавая резервные копии, старайтесь соблюдать следующие правила:

- **Правило 321** означает, что у вас всегда должно быть не менее *трех* копий данных, причем *две* из них должны находиться на двух разных носителях в разных местах, а *одна* — в другом географическом месте.

- **Проверяйте резервные копии.** Убеждайтесь, что каждая резервная копия содержит целостные данные и позволяет восстановить их, если понадобится. Это очень важное правило, которым часто пренебрегают.
- **Стратегии и процедуры резервного копирования** должны быть четко и полно документированы, чтобы все сотрудники могли использовать одни и те же методы.

В следующем разделе мы рассмотрим некоторые известные инструменты, с помощью которых можно создавать полные резервные копии системы Linux, — начиная с утилит, которые интегрированы в операционную систему, и заканчивая решениями сторонних разработчиков, которые одинаково хорошо подходят как для домашнего, так и для корпоративного использования.

Решения для клонирования дисков

Неплохой вариант резервного копирования — клонировать весь жесткий диск или несколько разделов, на которых хранятся важные данные. Для этой задачи Linux предлагает ряд многофункциональных инструментов, среди которых — команда `dd`, команда `ddrescue` и программа **ReaR**. В следующих разделах мы рассмотрим их подробнее.

Команда `dd`

Один из самых известных инструментов резервного копирования — команда `dd`, которую мы уже обсуждали в главе 6 «Как работать с дисками и файловыми системами». Давайте вспомним, как использовать ее в сценарии резервного копирования и восстановления. `dd` позволяет копировать отдельные блоки или даже байты из источника в приемник, независимо от типа файловой системы.

Давайте посмотрим, как клонировать весь диск. В нашей системе есть виртуальная машина с диском объемом 20 Гбайт, который мы хотим скопировать на накопитель USB объемом 128 Гбайт. Процедуры, которые мы сейчас продемонстрируем, будут работать с любыми дисками, а не только с виртуализированными.

Сначала давайте запустим команду `sudo fdisk -l`, чтобы проверить размеры всех дисков. В результате мы получим сведения о локальном диске и накопителе USB.

Теперь, когда мы знаем размеры дисков и убедились, что содержимое исходного диска (в нашем случае `/dev/vda`) уместится на целевой (`/dev/sda`), давайте его клонировать. Эта операция может занять некоторое время:

```
packt@neptune:~$ sudo dd if=/dev/vda of=/dev/sda conv=noerror,sync
status=progress
2245710336 bytes (2.2 GB, 2.1 GiB) copied, 175 s, 12.8 MB/s
```

В этой команде использовались такие ключи:

Ключ	Значение	Описание
if	/dev/vda	Входной файл (в нашем случае — исходный жесткий диск)
of	/dev/sda	Выходной файл (в нашем случае — целевой накопитель USB)
conv	noerror	Игнорировать ошибки чтения
	sync	В случае ошибок дополнять входные данные символами NUL до полных блоков, чтобы смещение данных всегда было синхронизировано
status	progress	Отображать статистику о процессе передачи данных

Имейте в виду, что эта операция может оказаться небыстрой; например, в нашей системе она заняла 200 минут. Мы сделали предыдущий листинг, когда копирование только началось. В следующем разделе мы покажем, как использовать `ddrescue`.

Команда `ddrescue`

`ddrescue` — еще один инструмент, с помощью которого можно клонировать диск. Эта утилита копирует данные с одного устройства или файла на другое, стараясь в первую очередь копировать только хорошие и здоровые участки. Если диск неисправен, имеет смысл запускать команду `ddrescue` дважды, поскольку в первый раз она скопирует только исправные участки, а сведения об ошибках перенесет в специальный файл отчета. Во второй раз будут копироваться только дефектные участки, поэтому для надежности лучше настроить команду на несколько попыток чтения.

В Ubuntu утилита `ddrescue` не установлена по умолчанию, но ее можно установить с помощью `apt`:

```
sudo apt install gddrescue
```

Мы будем использовать `ddrescue` на той же системе, что и в предыдущем примере, и клонировать тот же диск. Соответствующая команда выглядит так:

```
packt@neptune :~$ sudo ddrescue -n /dev/vda /dev/sda rescue.map --force
GNU ddrescue 1.27
Press Ctrl-C to interrupt
Current status
      ipos:   8800 MB, non-trimmed:      0 B,  current rate:   7340 kB/s
      opos:   8800 MB, non-scraped:      0 B,  average rate:  18258 kB/s
non-tried: 12674 MB, bad-sector:      0 B,   error rate:      0 B/s
   rescued:   8800 MB,  bad areas:      0,    run time:       8m  2s
pct rescued:  40.98%, read errors:      0,  remaining time:    7m
                                     time since last successful read:    0s
Copying non-tried blocks... Pass 1 (forwards)
Finished
```

Мы использовали ключ `--force`, чтобы перезаписать все данные на целевом диске. Эта операция тоже довольно длительная, поэтому будьте готовы подо-

ждать. В нашем случае клонирование заняло почти час. Далее мы продемонстрируем, как использовать еще один полезный инструмент — утилиту ReaR.

Утилита ReaR

ReaR (Relax-and-Recover¹) — это мощный инструмент аварийного восстановления и миграции систем, написанный на языке Bash. Он используется в дистрибутивах корпоративного класса, таких как RHEL и SLES, но его можно установить и в Ubuntu. Разработчики стремились к тому, чтобы ReaR было легко настраивать и использовать. Эту программу можно интегрировать с локальным загрузчиком, планировщиком `cron` или такими инструментами мониторинга, как **Nagios**. Чтобы получить более подробную информацию, посетите официальный сайт relax-and-recover.org.

Вот команда для того, чтобы установить ReaR в Ubuntu:

```
sudo apt install rear
```

После установки обратите внимание на главный конфигурационный файл `/etc/rear/local.conf`, в котором задаются все параметры программы. По умолчанию ReaR создает резервные копии в формате ISO, но также поддерживает такие целевые форматы, как Samba (CIFS), USB и NFS. Далее мы рассмотрим, как с помощью ReaR создать резервную копию данных на локальном сервере NFS.

Как создать резервную копию данных на локальном сервере NFS с помощью ReaR

В качестве примера мы продемонстрируем, как создавать резервные копии на сервере NFS. Чтобы выполнить это упражнение, в вашей сети должны быть доступны как минимум две системы: на одной будет установлен сервер NFS (в качестве сервера резервного копирования), а вторая будет выполнять роль рабочей машины, содержимое которой будет копироваться². На обеих системах должен быть установлен ReaR. Выполните следующие шаги:

1. Прежде всего нужно настроить сервер NFS. Подробно об этом рассказывается в главе 13 «Как настраивать серверы Linux», а здесь мы лишь вкратце затронем эту тему³. В конфигурационном файле сервера NFS `/etc/exports` хранятся сведения о том, какие ветви файловой системы нужно копировать.

¹ «Расслабиться и восстановиться». Это игра слов: в повседневной речи это устойчивое словосочетание относится к людям (когда идет речь, например, об отдыхе на курорте). — *Примеч. пер.*

² Например, если вы запускаете Ubuntu в виртуальной машине, то на этом этапе можно клонировать виртуальный образ и в одной системе установить сервер NFS, а в другой — ReaR. — *Примеч. ред.*

³ Если сервер NFS не установлен, его можно установить командой `sudo apt install nfs-kernel-server`. Затем стоит проверить его состояние (`sudo systemctl status nfs-kernel-server.service`) и, если окажется, что служба не запущена, запустить ее командой `sudo systemctl start nfs-kernel-server.service`. — *Примеч. ред.*

Прежде чем добавлять туда путь к нужному каталогу, создайте этот каталог. Мы будем размещать резервные копии NFS в каталоге `/home/export/`; давайте создадим внутри него подкаталог `rear`, где будут сохраняться копии, которые мы сформируем с помощью ReaR. Создать эти каталоги можно так:

```
sudo mkdir -p /home/export/rear
```

2. Теперь смените владельца целевого каталога. Если владельцем останется `root`, у ReaR не будет разрешений, чтобы записывать резервную копию в это место. Сменить владельца можно так:

```
sudo chown -R nobody:nogroup1 /home/export/rear
```

3. Закончив операции с каталогом, откройте файл `/etc/exports` в текстовом редакторе и добавьте новую строку для резервного каталога по такой схеме:
`/home/export/rear 192.168.124.0/24(rw, sync, no_subtree_check)`

Нашему компьютеру соответствует диапазон адресов `192.168.124.0/24`, но вы подставьте вместо него диапазон своей локальной сети².

4. Чтобы обновить конфигурацию сервера NFS, запустите такую команду:

```
sudo exportfs -a
```

Убедитесь, что новая конфигурация применилась:

```
sudo exportfs -v
```

Наконец, перезапустите службу NFS:

```
sudo systemctl restart nfs-kernel-server.service
```

5. После того как вы настроили сервер NFS, перейдите на машину, где установлен ReaR, и отредактируйте его конфигурационный файл `/etc/rear/local.conf`: добавьте в него строки, которые показаны ниже. По-прежнему вместо `192.168.124.0` укажите IP-адрес своего сервера NFS:

```
OUTPUT=ISO
```

```
# Формат образа восстановления (возможны также значения RAMDISK,
# PXE и др.)
```

```
OUTPUT_URL=nfs://192.168.124.0/home/export/rear
```

```
# Расположение образа восстановления
```

```
BACKUP=NETFS
```

```
# Стратегия резервного восстановления (NETFS — стратегия ReaR
# по умолчанию)
```

¹ Учетная запись `nobody` и группа `nogroup` — специальные идентификаторы Linux с «наименьшими» правами доступа. В среде реальной эксплуатации рекомендуется задавать владельца каталога более аккуратно — например, создать для этой цели специального пользователя и группу с тонко настроенными правами доступа. — *Примеч. ред.*

² Узнать IP-адрес и маску подсети системы можно, например, с помощью команды `nmcli` без аргументов. — *Примеч. ред.*


```
BACKUP_URL=nfs://192.168.124.0/home/export/rear  
# Расположение резервной копии в формате архива
```

6. Наконец, чтобы начать резервное копирование, запустите такую команду:

```
sudo rear -v -d mkbackup
```

Процедура копирования займет продолжительное время и выведет в терминал довольно крупный отчет, который мы не будем здесь показывать. После того как команда завершит работу, просмотрите каталог на сервере NFS и убедитесь, что в нем появилась резервная копия.

Если все прошло успешно, то в каталоге `/home/export/rear/` появится подкаталог с именем узла (в нашем случае `neptune`), а в этом каталоге — резервные копии и сопутствующие файлы. Файл `rear-neptune.iso` — это основная резервная копия, с помощью которой можно загрузить систему, если понадобится ее восстанавливать. Кроме того, в этом же каталоге появится архив `backup.tar.gz`, который содержит все файлы с нашей локальной машины.

Резервные копии ReaR именуются по определенной схеме: имя файла начинается с `rear-`, а затем следует имя узла системы и расширение `.iso`. Наша система называется `neptune`, поэтому в нашем случае файл резервной копии называется `rear-neptune.iso`.

После того как вы сохранили резервный образ ISO на сервере NFS, вы сможете восстановить систему с помощью накопителя USB или DVD с этим образом.

Как создать резервную копию данных на носителе USB с помощью ReaR

ReaR также позволяет создать резервную копию непосредственно на носителе USB. Вот как это можно сделать:

1. Вставьте носитель в порт USB и отформатируйте его с помощью такой команды (обратите внимание, что эта процедура может занять много времени):

```
packt@neptune:~$ sudo rear format /dev/sda1  
USB device /dev/sda is not formatted with ext2/3/4 or btrfs filesystem  
Type exactly 'Yes' to format /dev/sda with ext3 filesystem  
(default 'No' timeout 300 seconds)  
Yes2
```

¹ Носитель USB не обязательно представляется как `/dev/sda`. В зависимости от того, какие еще устройства подключены к системе, он может представляться как `/dev/sdb`, `/dev/sdc` и т. д. Узнать путь к носителю в вашей системе можно, например, с помощью команды `lsblk`. — *Примеч. ред.*

² Если эта команда почему-либо не сработает, можно отформатировать носитель USB вручную, как описывалось в разделе «Как создавать и форматировать разделы диска» главы 6. — *Примеч. ред.*

2. Теперь нужно изменить конфигурационный файл `/etc/rear/local.conf` так, чтобы резервные копии сохранялись на носителе USB. Вместо прежнего содержания введите в файл такие строки:

```
OUTPUT=USB
BACKUP_URL="usb:///dev/disk/by-label/REAR-000"
```

3. Чтобы разобраться в последней строке этого кода, можно запустить такую команду:

```
packt@neptune:~$ sudo ls -la /dev/disk/by-label/
total 0
drwxr-xr-x 2 root root 60 Apr 25 19:09 .
drwxr-xr-x 8 root root 160 Apr 25 19:09 ..
lrwxrwxrwx 1 root root 10 Apr 25 19:09 REAR-000 -> ../../sda1
```

4. Вывод показывает, что в нашем случае `/dev/disk/by-label/REAR-000` является ссылкой на `/dev/sda1`.
5. Чтобы создать резервную копию системы на носителе USB, запустите такую команду:

```
sudo rear -v mkbakup
```

Эта операция может выполняться довольно долго, поэтому наберитесь терпения. После этого на носителе USB появятся файлы резервной копии.

6. Чтобы восстановить систему, вам понадобится загрузиться с носителя USB и выбрать первый вариант Recover "*имя_узла*", где *имя_узла* — это имя компьютера, резервную копию которого вы создали.

Резервное копирование и восстановление системы — две очень важные функции, без которых не может обойтись ни один системный администратор Linux. Если вы умеете справляться с ними, это поможет сберечь данные, а также сэкономить время и деньги как для компании, так и для клиентов. Минимизировать время простоя, а также быстро и эффективно реагировать на нештатные ситуации — вот одни из самых приоритетных задач технических руководителей. Стратегии резервного копирования и восстановления всегда должны надежно опираться на передовые методы снижения рисков. Поэтому каждому системному администратору пригодится эффективный набор диагностических инструментов и знания о том, как устранять неполадки. В следующем разделе мы рассмотрим некоторые из лучших инструментов диагностики в Linux.

Популярные инструменты для диагностики и устранения неполадок в Linux

Одно из главных преимуществ Linux состоит в том, что это открытая экосистема, благодаря чему в ней развивается очень много решений для самых разных задач. Поэтому системным администраторам Linux доступен широкий выбор диагностических средств. Устранение неполадок, по сути, заключается в том,

что вы решаете проблемы на основе диагностики с помощью специальных инструментов. Чтобы не потеряться во всем их многообразии, в этом разделе мы сузим круг вопросов до таких тем:

- Неполадки загрузки.
- Неполадки потребления системных ресурсов.
- Неполадки сети.
- Неполадки оборудования.

Для каждой категории неполадок существуют свои диагностические инструменты, и сейчас мы рассмотрим наиболее распространенные из них.

Как устранять неполадки загрузки

Чтобы разобраться в том, какие проблемы могут возникнуть на этапе загрузки, важно знать, как происходит этот процесс. Мы еще не рассказывали об этом подробно, поэтому постарайтесь уделить должное внимание всему, о чем пойдет речь дальше.

Процесс загрузки

Во всех основных дистрибутивах Linux, таких как Ubuntu, OpenSUSE, Debian, Fedora и RHEL, в качестве загрузчика по умолчанию используется **GRUB2**, а в качестве подсистемы инициализации — **systemd**. Раньше, до того как эти системы стали общим стандартом, процесс загрузки Linux состоял из большого количества различных этапов.

Современный порядок загрузки устроен так:

1. Самотестирование при включении (POST) на уровне BIOS/UEFI.
2. Инициализация загрузчика GRUB2.
3. Инициализация ядра GNU/Linux.
4. Инициализация подсистемы инициализации **systemd**.

POST заключается в том, что BIOS или UEFI инициализирует и тестирует аппаратное обеспечение. Эта процедура одинакова для любого компьютера, независимо от того, какая операционная система на нем установлена и установлена ли она вообще. BIOS/UEFI проверяет, чтобы все аппаратные компоненты работали правильно. Если BIOS/UEFI не запускается, это обычно связано с тем, что какие-то компоненты неисправны или несовместимы друг с другом. Если POST проходит успешно, BIOS/UEFI ищет загрузочную запись диска, например **MBR** или **GPT**, и загружает ее в память.

Собственно Linux начинает работать с того момента, как инициализируется GRUB2. На этом этапе система загружает ядро Linux в память. Если на компьютере доступно несколько операционных систем, GRUB2 может выбирать между различными ядрами. После того как ядро загружено в память, оно управляет дальнейшим процессом загрузки.

Ядро — это самораспаковывающийся архив. После того как он извлечен, он выполняется в памяти и загружает подсистему инициализации, которая становится родителем (или предком более высокого уровня) всех остальных процессов в Linux.

Подсистема инициализации `systemd` начинает работу с того, что монтирует файловые системы и опрашивает все доступные конфигурационные файлы.

В процессе загрузки могут возникать проблемы. В следующем разделе мы расскажем, что делать, если случилась беда и загрузчик не запускается.

Как восстановить GRUB2

Если загрузчик GRUB2 выйдет из строя, вы не сможете получить доступ к системе, и загрузчик нужно будет восстанавливать. Здесь вам поможет загрузочный носитель USB. Рассматривайте дальнейшие шаги как учебное упражнение. Мы будем использовать «живой» (live) носитель Ubuntu 22.04 LTS (в настольной версии), но вы можете экспериментировать и с любым другим дистрибутивом Linux, если у вас есть соответствующий загрузочный носитель. Вот что нужно предпринять:

1. Подключите «живой» носитель с Ubuntu 22.04 и запустите систему.
2. Зайдите в BIOS/UEFI, выберите этот носитель в качестве основного загрузочного устройства и перезагрузите систему.
3. Выберите вариант Try Ubuntu (Попробовать Ubuntu).
4. Когда загрузится «живой» экземпляр Ubuntu, откройте Терминал, введите команду `sudo fdisk -l` и просмотрите диски и разделы.
5. Выберите раздел, на котором установлен GRUB2, и запустите команду такого вида (вместо указанных имен дисков используйте настоящие имена из своей системы):

```
sudo mount -t ext4 /dev/sda1 /mnt
```

6. Установите GRUB2 с помощью таких команд:

```
sudo chroot /mnt
grub-install /dev/sda
grub-install -recheck /dev/sda
update-grub
```

7. Размонтируйте раздел с помощью таких команд:

```
exit
sudo umount /mnt
```

8. Перезагрузите компьютер¹.

¹ Чтобы восстановить неисправный загрузчик, на практике чаще всего применяют загрузочный образ `boot-repair-disk` (sourceforge.net/projects/boot-repair-cd). — *Примеч. ред.*

С загрузчиками следует обращаться очень деликатно. Не упускайте из внимания мелкие детали и следите за тем, какие команды вы вводите, иначе все может пойти наперекосяк. В следующем разделе мы продемонстрируем некоторые диагностические инструменты, которые помогают решать общие системные проблемы.

Как устранять неполадки, связанные с потреблением системных ресурсов

Системные проблемы бывают разной природы и разной сложности, и крайне важно владеть инструментами, с помощью которых можно решать эти проблемы. В этом разделе мы рассмотрим средства, которые входят в дистрибутивы Linux по умолчанию. Элементарные навыки устранения неполадок необходимы любому системному администратору Linux, потому что во время повседневной работы обязательно будут происходить какие-то проблемы.

Что мы подразумеваем под общими системными неполадками? В основном это проблемы с дисковым пространством, использованием памяти, загрузкой системы и запущенными процессами.

Как устранять неполадки с дисковым пространством

Дисковые устройства, будь то HDD или SSD, играют важную роль в системе. Они обеспечивают необходимое пространство для данных, файлов и всевозможного программного обеспечения, включая операционную систему. Мы пока не будем обсуждать проблемы, связанные с аппаратным обеспечением, а отложим их до завершающего раздела этой главы — «Как устранять неполадки оборудования». А здесь мы рассмотрим проблемы, которые касаются **дискового пространства**. Соответствующие диагностические инструменты уже установлены в любой системе Linux и представлены такими командами:

du	Показывает, сколько места занимает тот или иной файл или каталог
df	Показывает общую картину того, как используется дисковое пространство, и позволяет узнать, какие файлы и каталоги занимают больше всего места

Вот пример запуска df с ключом -h (--human-readable, человекочитаемый вывод). Для краткости здесь приводятся только избранные строки вывода:

```
packt@neptune:~$ df -h
Filesystem      Size  Used Avail Use% Mounted on
tmpfs            336M  1.5M  334M   1% /run
efivarfs        256K   63K  189K  25% /sys/firmware/efi/efivars
/dev/mapper/ubuntu--vg-ubuntu--lv 9.8G  2.5G   6.8G  27% /
tmpfs            1.7G     0   1.7G   0% /dev/shm
... и т. д. ...
tmpfs            336M  12K  336M   1% /run/user/1000
```

Если на одном из дисков закончится место, команда сообщит об этом. В нашем примере такой проблемы нет, но с помощью `df` все равно можно отследить, на каких дисках остается мало свободного пространства.

Когда диск заполнен (или почти заполнен), ситуацию можно исправить разными способами. Если вам придется удалять какие-то файлы, удаляйте их прежде всего из каталога `/home` и старайтесь не трогать важные системные файлы. Удаляя файлы, попробуйте такие стратегии:

- Удалите ненужные файлы с помощью команды `rm` (в частности, с ключами `-rf`, если вы уверены в том, что делаете) или `rmdir`.
- Переместите файлы на внешний диск (или в облако) с помощью команды `rsync`.
- Найдите подкаталоги, которые занимают больше всего места в каталоге `/home`.

Вот пример того, как с помощью утилиты `du` можно найти самые большие каталоги внутри каталога `/home`¹. Мы используем трехступенчатый конвейер: сначала вывод команды `du` передается в команду `sort`, а затем уже ее вывод — в команду `tail` с ключом `+2` (чтобы исключить сам каталог `/home`), после чего список каталогов передается в `head`, чтобы оставить только пять самых больших каталогов:

```
packt@neptune:~$ sudo du -h /home | sort -rh | tail +2 | head -5
181M    /home/packt
16K     /home/alex
12K     /home/packt/reports
12K     /home/packt/local
```

В других случаях, чтобы устранять неполадки, нас интересует не столько дисковое пространство, сколько количество используемых индексных дескрипторов. Чтобы проверить, не исчерпались ли они, можно использовать команду `df -i`:

```
packt@neptune:~$ df -i
Filesystem                                Inodes   IUsed IFree  IUse% Mounted on
tmpfs                                     252890    810 252080    1% /run
/dev/mapper/ubuntu--vg-ubuntu--lv        655360 120212 535148   19% /
tmpfs                                     252890     1 252889    1% /dev/shm
tmpfs                                     252890     3 252887    1% /run/lock
tmpfs                                     50578     25 50553    1% /run/user/1000
```

Этот вывод показывает основные сведения о том, как используются индексные дескрипторы. Вы видите их общее количество в различных файловых системах, а также сколько дескрипторов используется (в абсолютном и процентном выражении), а сколько остаются свободными.

Помимо перечисленных команд, которые входят по умолчанию в любой дистрибутив Linux, существует много других инструментов с открытым исходным кодом,

¹ В большинстве случаев «виновником» является переполнение каталога `/var/log`. По этой причине на этапе установки операционной системы его выносят на отдельный раздел. — *Примеч. ред.*

которые помогают решать проблемы с дисковым пространством, например `pydf`, `parted`, `sfdisk`, `iostat` и приложение **GParted** с графическим интерфейсом.

В следующем разделе мы рассмотрим команды, которые позволяют проверять возможные проблемы с памятью.

Как устранять неполадки с потреблением памяти

Потребление памяти — один из важных факторов общей производительности системы, наряду с загрузкой процессора и использованием диска. Очень важно проверять потребление системных ресурсов с помощью специальных инструментов. Для мониторинга оперативной памяти в Linux по умолчанию используется утилита `free`, которая доступна во всех основных дистрибутивах. В следующем примере мы запускаем ее с ключом `-h`, чтобы получить вывод в человекочитаемом формате:

```
packt@neptune:~$ free -h
              total        used          free      shared  buff/cache   available
Mem:          1.9Gi         251Mi         745Mi         1.0Mi         978Mi         1.5Gi
Swap:          1.8Gi           0.0Ki         1.8Gi
```

Эта команда выводит сведения об отдельных категориях памяти:

total	Общий объем памяти
used	Память, которая используется или недоступна по другим причинам
free	Свободная память
shared	Память, которую использует временное хранилище <code>tmpfs</code>
buff/cache	Память, которую используют буферы ядра и страничный кэш
available	Память, доступная для новых приложений

Команда `free` часто помогает выявить конкретные проблемы, связанные с перерасходом памяти. На действующих серверах важно постоянно следить за тем, как потребляется память, чтобы вовремя заметить соответствующие неполадки.

Проверить использование памяти можно также с помощью команды `top`, как показано ниже:

```
packt@neptune:~$ top
top - 19:45:13 up 1:37, 2 users, load average: 0.00, 0.00, 0.06
Tasks: 135 total, 1 running, 134 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.0 us, 0.0 sy, 0.0 ni, 98.7 id, 1.3 wa, 0.0 hi, 0.0 si, 0.0 st
MiB Mem : 1975.7 total, 745.3 free, 251.2 used, 979.2 buff/cache
MiB Swap: 1840.0 total, 1839.7 free, 0.3 used. 1534.1 avail Mem

  PID USER  PR  NI   VIRT   RES   SHR S  %CPU  %MEM    TIME+  COMMAND
4157 root   20   0      0      0      0 I    0.3   0.0 0:00.15 kworker/0:2-events
1 root    20   0 167828 13072 8028 S    0.0   0.6 0:01.74 systemd
2 root    20   0      0      0      0 S    0.0   0.0 0:00.00 kthreadd
... и т. д. ...
```

Команда `top` выводит данные в нескольких столбцах, которые уже рассматривались в разделе «Как устроены процессы» главы 5. Данные отображаются в динамическом режиме: они непрерывно обновляются, отражая сведения о запущенных процессах в реальном времени. В частности, столбец `RES` показывает объем занимаемой памяти в мегабайтах, а `%MEM` — в процентах.

Команда `vmstat` выводит сведения о виртуальной памяти, а также другую важную системную информацию:

```
packt@neptune :~$ vms t at
procs -----memory----- ---swap-- -----io----- -system-- -----cpu----
 r b swpd free buff cache si so bi bo in cs us sy id wa st
 1 0 268 763164 552328 450412 0 0 1616 1418 511 1045 0 1 85 14 0
```

По умолчанию `vmstat` показывает избранные сведения о процессах, виртуальной памяти, файле подкачки, подсистеме ввода-вывода и использовании процессора. Информация о виртуальной памяти содержит такие столбцы:

<code>swpd</code>	Сколько виртуальной памяти используется
<code>free</code>	Сколько памяти свободно
<code>buff</code>	Сколько памяти используется для буферизации
<code>cache</code>	Сколько памяти используется для кэширования

У команды `vmstat` есть много полезных ключей. Чтобы узнать об этих ключах, а также обо всех доступных столбцах выходных данных, обратитесь к соответствующей справочной странице (`man vmstat`).

Если запустить команду `vmstat` с ключом `-a` (`--active`), она выведет объем активной и неактивной памяти¹:

```
packt@neptune:~$ vmstat -a
procs -----memory----- ---swap-- -----io----- -system-- -----cpu----
 r b swpd free inact active si so bi bo in cs us sy id wa st
 2 0 268 762912 700560 301296 0 0 1589 1393 503 1028 0 1 86 13 0
```

Команда `vmstat` с ключом `-s` (`--stats`) выводит подробные сведения о состоянии файла подкачки и операциях с ним.

Все команды, которые рассматривались в этом разделе, крайне полезны для того, чтобы устранять неполадки с памятью, и вы найдете эти команды по умолчанию в любом дистрибутиве Linux.

Тем не менее стоит упомянуть еще одну команду — `sar`², которую можно установить в Ubuntu в составе пакета `sysstat`:

```
sudo apt install sysstat
```

¹ Память относится к *активной*, если она используется теми или иными действующими процессами; иначе память считается *пассивной*. — *Примеч. ред.*

² Сокр. System Activity Reporter (генератор отчетов системной активности). — *Примеч. пер.*

Эта команда выводит подробную статистику того, как используются ресурсы компьютера, в том числе процессор, оперативная память, сеть и даже электропитание. Чтобы она работала, нужно включить службу `sysstat`:

```
sudo systemctl enable sysstat
sudo systemctl start sysstat
```

По умолчанию эта служба запускается каждые 10 минут и сохраняет журналы в файлах `/var/log/sysstat/saXX`, где `XX` — текущая дата. (Например, мы запускаем команду `sar` 25 апреля, и соответствующие журналы хранятся в файле `/var/log/sysstat/sa25`).

С помощью `sar` можно генерировать различные отчеты в режиме реального времени. Например, если мы хотим сгенерировать 5 отчетов о памяти через каждые 2 секунды, можно использовать ключ `-r` таким образом (для краткости здесь показаны только первые 7 столбцов вывода):

```
packt@neptune:~$ sudo sar -r 2 5
Linux 5.15.0-70-generic (neptune) 04/25/2023 _x86_64_ (2 CPU)
07:52:29 PM kbmemfree  kbavail  kbmempused  %memused  kbbuffers  kbcached
07:52:31 PM      828648    1564124     219728      10.86      491816     338824
07:52:33 PM      828648    1564124     219732      10.86      491816     338824
07:52:35 PM      828648    1564124     219732      10.86      491824     338816
07:52:37 PM      828648    1564124     219724      10.86      491824     338824
07:52:39 PM      828648    1564124     219728      10.86      491824     338824
Average:         828648    1564124     219726      10.86      491821     338822
```

Данные, которые выводит `sar`, собирает в фоновом режиме утилита `sadc`¹, которая тоже входит в пакет `sysstat`.

Даже если все предыдущие команды сработали благополучно, после перезагрузки системы служба `sysstat` может не перезапуститься по умолчанию. Чтобы решить эту проблему, отредактируйте файл `/etc/default/sysstat` и измените значение `ENABLED` с `false` на `true`.

В этом разделе мы рассмотрели самые важные инструменты, которые позволяют устранять неполадки с памятью. В следующем разделе поговорим о том, как решать проблемы с загруженностью системы.

Как устранять неполадки с нагрузкой на систему

В этом разделе мы обсудим проблемы с **нагрузкой на систему**. Чтобы их диагностировать, можно использовать инструменты, которые мы уже рассматривали. Например, команда `top` становится одной из самых востребованных, когда мы пытаемся выяснить, почему компьютер «тормозит». Все остальные инструменты — например, `vmstat` и `sar`, — тоже помогают устранять неполадки с нагрузкой на процессор и систему.

¹ Сокр. System Activity Data Collector (сборщик данных о системной активности). — Примеч. пер.

Самая общая команда, которая позволяет диагностировать нагрузку на систему, называется `uptime`:

```
packt@neptune:~$ uptime
20:01:48 up 1:53, 2 users, load average: 0.00, 0.00, 0.00
```

В конце вывода показаны три значения `load average` (средняя загруженность) — это средние показатели нагрузки на процессор¹ за последние 1, 5 и 15 минут. Эти значения позволяют получить достоверное представление о том, что происходит с процессами в системе.

Если в вашей системе один процессор, то среднюю загруженность **1** можно трактовать так, что он задействуется полностью. Если этот показатель еще выше, это значит, что многие процессы ожидают своей очереди на выполнение, отчего снижается общая производительность системы.

Высокая средняя загруженность может означать, что некоторые приложения запускают несколько потоков одновременно. Впрочем, некоторые проблемы с загруженностью возникают не только из-за того, что не хватает ресурсов процессора; они могут быть результатом комбинированной нагрузки на процессор, дисковый ввод-вывод и память. В этом случае универсальным диагностическим инструментом служит команда `top`, вывод которой постоянно меняется в реальном времени в зависимости от состояния системы.

По умолчанию `top` сортирует процессы по тому, насколько активно они потребляют процессор. Эта команда работает в интерактивном режиме, в котором не всегда удастся уследить за выводом на экране. С помощью ключа `-b (--batch)` можно запустить `top` в пакетном режиме, в котором она не взаимодействует с пользователем, а только обновляет вывод в зависимости от того, какое количество итераций указано при ключе `-n (--iterations)`. Вывод `top` в пакетном режиме можно перенаправить в файл — например, с помощью `tee`:

```
packt@neptune:~$ top -b -n 1 | tee top-command-output
top - 15:20:30 up 3 min, 2 users, load average: 0.21, 0.17, 0.07
Tasks: 154 total, 1 running, 153 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.0 us, 0.0 sy, 0.0 ni, 100.0 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
MiB Mem: 3911.9 total, 1135.6 free, 244.9 used, 2531.5 buff/cache
MiB Swap: 1840.0 total, 1840.0 free, 0.0 used. 3413.7 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1	root	20	0	167740	13212	8256	S	0.0	0.3	0:02.94	systemd
2	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kthreadd
3	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	rcu_gp
4	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	rcu_par_gp
5	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	slub_flushwq

¹ Эти показатели вычисляются как среднее количество процессов, которые выполняются на процессоре, ожидают выполнения или находятся в состоянии непрерываемо-го ожидания. — *Примеч. ред.*

Вот что означают поля вывода, которые касаются процессорного времени:

Поле	Процессорное время, затраченное на...
us	Пользовательские процессы со стандартным приоритетом
sy	Процессы ядра
ni	Процессы с нестандартным приоритетом (например, измененным с помощью команды <code>nice</code>)
id	Простой
wa	Ожидание ввода-вывода
hi	Обслуживание аппаратных прерываний
si	Обслуживание программных прерываний
st	Ожидание виртуальной машиной доступа к ресурсу физического процессора

Еще один полезный инструмент, с помощью которого можно изучать загруженность процессора и время ввода-вывода данных для различных устройств, — утилита `iostat`:

```
packt@neptune:~$ iostat
Linux 5.15.0-70-generic (neptune)      04/25/2023      _x86_64_      (2 CPU)

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           0.10    0.01   0.82   11.40    0.01   87.67

Device            tps    kB_read/s    kB_wrtn/s    kB_dscd/s    kB_read    kB_wrtn    kB_dscd
dm-0               8.14       78.94       39.11       446.81    558397    276624    3154984
loop0              0.05        0.16        0.00        0.00       1098         0         0
loop1              0.05        0.16        0.00        0.00       1113         0         0
... и т. д. ...
loop9              0.01        0.07        0.00        0.00        502         0         0
sda               155.27      616.49     2329.90        0.00   4360936  16481248         0
vda                21.32      2068.83      39.13      446.01  14634494   276780    3154984
```

Статистика процессора (`avg-cpu`) отображает данные того же характера, что мы обсуждали ранее в связи с командой `top`. Ниже показана статистика ввода-вывода для различных устройств, которая состоит из таких столбцов¹:

tps	Количество запросов ввода-вывода в секунду
kB_read/s	Скорость чтения данных с устройства (Кбайт/с)
kB_wrtn/s	Скорость записи данных на устройство (Кбайт/с)
kB_dscd/s	Скорость отбраковывания данных для устройства (Кбайт/с)
kB_read	Общий объем прочитанных данных (Кбайт)
kB_wrtn	Общий объем записанных данных (Кбайт)
kB_dscd	Общий объем отброшенных данных (Кбайт)

¹ По умолчанию `iostat` выводит данные за период с последней загрузки системы. — *Примеч. ред.*

Чтобы подробнее узнать о команде `iostat`, обратитесь к соответствующей справочной странице (`man iostat`).

Еще одна полезная команда — `iotop`, которая отображает статистику ввода-вывода для отдельных процессов и потоков. Она не установлена по умолчанию в Ubuntu, но ее можно установить с помощью такой команды:

```
sudo apt install iotop-c
```

Чтобы запустить `iotop`, понадобятся права `sudo`:

```
sudo iotop
```

Проблемы с загруженностью системы также помогает решить служба `sysstat`, которую мы уже использовали, когда устраняли неполадки с памятью.

По умолчанию `sar` выводит статистику процессора за текущую дату:

```
packt@neptune:~$ sar 2 5
Linux 5.15.0-70-generic (neptune)      04/25/2023      _x86_64_      (2 CPU)

06:09:59 PM   CPU      %user   %nice   %system   %iowait   %steal   %idle
06:10:01 PM   all       0.00    0.00    0.50     0.00     0.00    99.50
06:10:03 PM   all       0.00    0.00    0.00     0.00     0.00   100.00
06:10:05 PM   all       0.00    0.00    0.00     0.00     0.00   100.00
06:10:07 PM   all       0.00    0.00    0.00     0.00     0.00   100.00
06:10:09 PM   all       0.00    0.00    0.25     0.00     0.00    99.75
Average:      all       0.00    0.00    0.15     0.00     0.00    99.85
```

В этом примере команда `sar` запускалась пять раз через каждые две секунды. Текущий компьютер не испытывает большой нагрузки, но попробуйте представить себе, каков может быть результат, если запустить команду на высоконагруженном сервере. Как мы уже упоминали в предыдущем разделе, у команды `sar` есть несколько ключей, которые часто помогают диагностировать и решать проблемы. Запустите `man sar`, чтобы просмотреть соответствующую справочную страницу.

Существует множество других инструментов, с помощью которых тоже можно искать и устранять неполадки, связанные с использованием системных ресурсов. В этом разделе мы едва затронули самые распространенные средства этого профиля. Разыскивайте и осваивайте новые инструменты, если почувствуете такую необходимость. Впрочем, в большинстве случаев тех утилит, которые рассматривались в этом разделе, будет достаточно, чтобы формировать полезные отчеты о возможных проблемах системы.

В следующем разделе мы поговорим о том, как устранять неполадки сетевого характера.

Как устранять неполадки, связанные с сетью

Довольно часто неполадки объясняются сложностями, которые присущи компьютерным сетям. Мы используем их повсюду: от умных часов до смартфонов,

компьютеров и облачных хранилищ. С помощью сетей все в мире связано со всем, отчего жизнь простых пользователей становится удобнее, а работа системных администраторов усложняется. В таком многосвязном мире то и дело что-нибудь идет не так, и приходится устранять сетевые неполадки.

По нашим наблюдениям из практики, устранение неполадок в **сети** составляет примерно 80 % работы системного администратора, а то и больше. Поскольку большинство неполадок серверов и облачных служб сводится к сетевым проблемам, стоит поддерживать сеть в исправном состоянии, чтобы время простоя сокращалось, а клиенты и системные администраторы оставались довольными.

Инструменты, которые мы обсудим в этом разделе, используются по умолчанию во всех основных дистрибутивах Linux. Все они рассматривались в главе 7 «Сетевое окружение Linux» и главе 9 «Как обеспечивать безопасность в Linux» или будут рассматриваться в главе 13 «Как настраивать серверы Linux», поэтому здесь мы поговорим о них только в контексте устранения неполадок. Давайте по отдельности рассмотрим инструменты, которые относятся к разным уровням стека TCP/IP. Вы наверняка помните, что в модели TCP/IP всего пять уровней, и мы проследуем по ним снизу вверх. Как показывает практика, искать и устранять неполадки в сети лучше всего по всему стеку.

Диагностика физического уровня (уровень 1)

Один из самых популярных инструментов тестирования и один из первых, которым пользуется большинство системных администраторов, — это команда `ping`, которая элементарными методами проверяет целостность и качество соединения¹. Давайте с нашего локального компьютера запросим один из известных доменных адресов, чтобы убедиться, что между этими узлами есть связь. Запустим проверку четыре раза, используя ключ `-c`:

```
packt@neptune:~$ ping -c 4 google.com
PING google.com (216.58.211.14) 56(84) bytes of data.
64 bytes from muc03s13-in-f14.1e100.net (216.58.211.14): icmp_seq=1 ttl=128
time=79.3 ms
64 bytes from muc03s13-in-f14.1e100.net (216.58.211.14): icmp_seq=2 ttl=128
time=24.3 ms
64 bytes from muc03s13-in-f14.1e100.net (216.58.211.14): icmp_seq=3 ttl=128
time=24.5 ms
64 bytes from muc03s13-in-f14.1e100.net (216.58.211.14): icmp_seq=4 ttl=128
time=24.3 ms

--- google.com ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3004ms
rtt min/avg/max/mdev = 24.334/38.121/79.346/23.800 ms
```

¹ Майк Муусс, который разработал первую версию программы `ping` в 1983 году, дал ей такое название в качестве подражания звукам, которые издает гидролокатор. Впоследствии также распространился бэкronym Packet InterNet Groper (можно примерно перевести как «ощупыватель интернет-пакетов»). — *Примеч. ред.*

Команда `ping` отправляет простые пакеты ICMP по адресу назначения (в нашем случае `google.com`) и ждет ответа. Если ответ получен и пакеты не потеряны, это означает, что обмен данными состоялся успешно. С помощью `ping` можно проверять соединения как с узлами в локальной сети, так и с удаленными узлами. Если нужно выявить возможные проблемы, чаще всего начинают с этого инструмента.

Бывают случаи, когда простого опроса с помощью `ping` недостаточно. В этом случае может помочь еще одна универсальная команда — `ip`, с помощью которой можно проверить, нет ли проблем на физическом уровне, например:

```
packt@neptune:~$ ip link show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT
group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
mode DEFAULT group default qlen 1000
    link/ether 00:0d:2f:0d:12:4d brd ff:ff:ff:ff:ff:ff
    altname enp2s1
```

Этот вывод показывает, что интерфейс Ethernet работает нормально (`state UP`). На компьютере, который мы использовали для этого примера, нет беспроводного соединения, но если бы оно было, то отобразилось бы под именем вроде `wlp0s20f3`.

Если бы какой-то из интерфейсов не работал, то в выводе предыдущей команды для него было бы отражено состояние `DOWN`. В этом случае можно попробовать «поднять» его с помощью такой команды (вместо `wlp0s20f3` подставьте имя настоящего интерфейса):

```
ip link set wlp0s20f3 up
```

После этого стоит снова проверить состояние интерфейса, запустив `ip link show`.

Если у вас есть прямой доступ к физической системе — например, серверу, — вы можете напрямую проверить, подключены ли провода. Правда, если вы используете беспроводное соединение (что не рекомендуется), вам придется запускать команду `ip`.

Еще один полезный инструмент для диагностики на уровне 1 — `ethtool`, который установлен по умолчанию в Ubuntu 22.04.2 LTS. Чтобы проверить интерфейс Ethernet, выполните следующую команду (здесь `ens33` — имя интерфейса, которое вывела команда `ip link`):

```
ethtool ens33
```

С помощью `ethtool`, в частности, можно проверить, правильно ли настроена скорость соединения. Например, следующие строки говорят о том, что система правильно согласовала полнодуплексное соединение на 1000 Мбит/с:

```
Speed: 1000Mb/s
Duplex: Full
Auto-negotiation: on
```

В следующем разделе мы покажем, как диагностировать второй уровень стека.

Диагностика канального уровня (уровень 2)

Второй уровень в стеке ТСП/IP называется **канальным** и обычно отвечает за подключение к локальной сети. Большинство проблем, которые могут возникнуть на этом этапе, связаны с неправильным сопоставлением IP- и MAC-адресов. Решать эти проблемы часто помогают такие инструменты, как `ip` и `arp`. Команда `arp` позволяет управлять соответствием IP-адресов (уровень 3) и MAC-адресов (уровень 2). В Ubuntu `arp` доступна в пакете `net-tools`, который можно установить с помощью такой команды:

```
sudo apt install net-tools
```

Чтобы просмотреть текущую таблицу сопоставлений ARP для IPv4, можно вызвать команду `arp` без ключей, например:

```
packt@neptune:~$ arp
Address                  HWtype  HWaddress           Flags Mask            Iface
192.168.126.1            ether    00:50:56:c0:00:08    C                    ens33
192.168.126.254          ether    00:50:56:ec:f1:be    C                    ens33
_gateway                 ether    00:50:56:f1:c5:6f    C                    ens33
```

Вывод этой команды показывает все подключенные устройства с подробной информацией об их IP- и MAC-адресах.

Более современным аналогом команды `arp` считается `ip neighbor`, вывод которой показан далее:

```
packt@neptune:~$ ip neighbour
192.168.126.1 dev ens33 lladdr 00:50:56:c0:00:08 DELAY
192.168.126.254 dev ens33 lladdr 00:50:56:ec:f1:be STALE
192.168.126.2 dev ens33 lladdr 00:50:56:f1:c5:6f STALE
```

Вместо подкоманды `neighbor` можно ввести `neighbour` или просто `neigh` — все три варианта работают одинаково. С помощью `ip neighbor` можно удалять записи из кэша сопоставления ARP, например:

```
ip neighbor delete 192.168.126.2 ens33
```

Команды `arp` и `ip neighbor` во многом похожи; они обе довольно мощные и обе помогают диагностировать возможные проблемы на уровне 2. В следующем разделе мы покажем, как диагностировать третий уровень стека.

Диагностика уровня интернета (уровень 3)

На уровне интернета (уровне 3) мы работаем только с IP-адресами. Мы уже знаем, какие инструменты здесь используются: это команды `ip`, `ping`, `traceroute` и `nslookup`.

Чтобы посмотреть текущую таблицу маршрутизации и список шлюзов для различных маршрутов, можно запустить команду `ip route show`:

```
packt@neptune: ~ $ ip route show
default via 192.168.124.1 dev enp1s0 proto dhcp src 192.168.124.112 metric 100
192.168.124.0/24 dev enp1s0 proto kernel scope link src 192.168.124.112 metric 100
192.168.124.1 dev enp1s0 proto dhcp scope link src 192.168.124.112 metric 100
```

В первой строке вывода команда `ip route show` показывает шлюз по умолчанию. Иногда проблема может заключаться в том, что он отсутствует или неправильно настроен.

Команда `traceroute` не установлена по умолчанию в Ubuntu, но ее можно установить обычным образом:

```
sudo apt install traceroute
```

Эта команда позволяет проследить путь трафика от отправителя к получателю. Здесь показано, по какому маршруту проходят пакеты от нашего локального шлюза до серверов Google:

```
packt@neptune:~$ traceroute google.com
traceroute to google.com (142.250.74.46), 64 hops max
 1  192.168.126.2 (_gateway)  0.101ms  0.211ms  0.058ms
 2  10.4.22.2 (10.4.22.2)  40.305ms  41.272ms  63.035ms
 3  91.236.116.1 (91.236.116.1)  68.169ms  40.067ms  62.123ms
 4  195.128.255.2 (et-0-0-23.kis-kg-4.obe.net)  41.486ms  66.189ms  59.563ms
 5  195.128.254.76 (ae0-8.kis-kg.obe.net)  87.167ms  21.272ms  40.976ms
 6  195.128.254.12 (ae1-354.bmm-sk1.obe.net)  61.913ms  80.547ms  61.210ms
 7  195.128.255.63 (et0-366.bmm-sk1-2.obe.net)  41.606ms  41.815ms  42.061ms
 8  142.250.174.204 (142.250.174.204)  61.281ms  41.124ms  88.740ms
 9  142.251.246.197 (142.251.246.197)  100.266ms  25.321ms  99.488ms
10  142.250.239.183 (142.250.239.183)  21.467ms  21.512ms  72.808ms
11  142.250.74.46 (arn09s22-in-f14.1e100.net)  43.056ms  84.188ms  90.787ms
```

Когда пакеты передаются в обратном направлении от получателя к отправителю, они обычно следуют не по тому же самому маршруту. Пакеты проходят через шлюзы, где обрабатываются и маршрутизируются. Когда данные выходят за пределы локальной сети, `traceroute` может неточно отразить их маршрут, потому что многие промежуточные шлюзы отфильтровывают соответствующие пакеты (например, пакеты ICMP с истекшим сроком годности).

Есть также более современный инструмент `tracpath`, похожий на `traceroute`. Если он не установлен в вашей системе по умолчанию, его можно установить в составе пакета `iputils-tracpath`. Обе утилиты по умолчанию отслеживают прохождение пакетов через порты UDP. `tracpath` с ключом `-n` выводит IP-адреса вместо имен узлов, а с ключом `-b` — как имена, так и адреса. Вот пример запуска этой команды:

```
packt@neptune:~$ tracpath 184.95.56.34
1?: [LOCALHOST]                                pmtu 1500
1:  10.240.134.2                                2.329ms
1:  10.240.134.2                                2.053ms
2:  sw1.edge.blgrd.srb2.cwie.net                 9.837ms
3:  10.234.5.2                                   2.395ms
```



```

4: 131.153.98.9 2.242ms
5: te0-2-0-7-0.ccr51.beg03.atlas.cogentco.com 2.677ms
6: be3422.ccr31.bud01.atlas.cogentco.com 8.156ms
7: be3261.ccr21.bts01.atlas.cogentco.com 10.366ms
8: be2988.ccr51.vie01.atlas.cogentco.com 11.554ms
9: be3462.ccr22.muc03.atlas.cogentco.com 16.817ms
10: be2960.ccr42.fra03.atlas.cogentco.com 22.771ms
11: be2800.ccr42.par01.atlas.cogentco.com 31.998ms
12: be2315.ccr31.bio02.atlas.cogentco.com 44.418ms
13: be2332.ccr42.dca01.atlas.cogentco.com 123.869ms asymm 15
14: be3084.ccr41.iad02.atlas.cogentco.com 124.888ms asymm 16
15: be2746.rcr21.b023801-0.iad02.atlas.cogentco.com 112.367ms
16: 38.88.249.10 115.346ms
17: no reply
18: no reply
19: 10.110.10.1 159.131ms asymm 20
20: eth.14.2.cr2.phx0.phoenixnap.com 162.346ms asymm 21
21: 10.220.60.10 159.480ms asymm 22
22: speedtest.phoenixnap.com 159.507ms !H
Resume: pmtu 1500

```

Еще одной сетевой проблемой может оказаться неправильное разрешение DNS, когда доступ к узлам удастся получить только по IP-адресам, а не по именам. Чтобы диагностировать подобные неполадки, даже если дело не в протоколе уровня 3, можно использовать команду `nslookup` в сочетании с `ping`.

Если `nslookup` выводит тот же IP-адрес, что и `ping`, значит, все в порядке, но если адрес отличается — значит, проблема в конфигурации узла. В следующем разделе мы рассмотрим, как диагностировать неполадки на уровнях 4 и 5.

Диагностика транспортного и прикладного уровней (уровни 4 и 5)

Верхние два уровня — транспортный (уровень 4) и прикладной (уровень 5) — в основном предоставляют каналы связи, через которые приложения на разных узлах обмениваются данными. Мы рассмотрим эти два уровня лишь в общих чертах. Два наиболее известных протокола транспортного уровня — **TCP** и **UDP**, которые реализованы в любой операционной системе и в совокупности охватывают весь трафик в интернете.

Один из важных инструментов, с помощью которого можно устранять неполадки на уровне 4, — команда `ss`¹. Она считается более современной заменой `netstat` и позволяет просматривать список всех сетевых сокетов. Этот список может быть довольно объемным, но с помощью определенных ключей его можно отфильтровать.

Например, ключ `-t` позволяет просматривать только сокет TCP, ключ `-u` — сокет UDP, а ключ `-x` — только сокет Unix. Таким образом, чтобы увидеть информацию о сокетах TCP и UDP, нужно запустить команду `ss` с ключами `-tu`. А чтобы увидеть все прослушиваемые сокеты в системе, можно использовать

¹ Сокр. socket statistics (статистика сокетов). — *Примеч. пер.*

ключ -l. Таким образом, все три ключа в сочетании покажут вам все сокеты UDP и TCP, которые прослушиваются в системе:

```
packt@neptune:~$ ss -utl
Netid State Recv-Q Send-Q Local Address:Port Peer Address:Port
Process
udp UNCONN 0 0 127.0.0.54:domain 0.0.0.0:*
udp UNCONN 0 0 127.0.0.53%lo:domain 0.0.0.0:*
udp UNCONN 0 0 10.255.255.254:domain 0.0.0.0:*
udp UNCONN 0 0 127.0.0.1:323 0.0.0.0:*
udp UNCONN 0 0 [::1]:323 [::]:*
tcp LISTEN 0 4096 127.0.0.54:domain 0.0.0.0:*
tcp LISTEN 0 1000 10.255.255.254:domain 0.0.0.0:*
tcp LISTEN 0 4096 127.0.0.53%lo:domain 0.0.0.0:*
```

В частности, команда `ss` помогает искать неполадки в сети, когда необходимо опросить сокеты, которые находятся в определенном состоянии, например `TIME_WAIT` (это сокеты, которые разорвали соединение). Для этого можно использовать такую команду (ключ `-o` здесь выводит таймеры для каждого сокета):

```
packt@neptune:~$ ss -o state time-wait
Netid Recv-Q Send-Q Local Address:Port Peer Address:Port Process
icmp6 0 0 *%ens33:ipv6-icm p *.*
timer: (timewait,227ms,0)
```

Подробнее о состоянии `TIME_WAIT` можно прочесть на странице vincent.bernat.ch/en/blog/2014-tcp-time-wait-state-linux¹.

Устраняя сетевые неполадки, всегда держите утилиту `ss` под рукой. На прикладном уровне (уровень 5) работают протоколы пользовательских приложений — например, **DHCP**, **HTTP** и **FTP**. Поскольку диагностика на уровне 5 состоит в основном в том, чтобы разбираться в неисправностях приложений, в этом разделе мы не будем ее рассматривать.

В заключительном разделе этой главы мы поговорим о том, как устранять неполадки с оборудованием.

Как устранять неполадки оборудования

Если вы решаете аппаратные проблемы, стоит начать с того, чтобы проверить оборудование. Хорошим инструментом для этого служит команда `dmidecode`, которая предоставляет подробные сведения о каждом аппаратном компоненте в человекочитаемом формате. Каждый элемент аппаратного обеспечения относится к определенному типу по классификации DMI², на которую опирается

¹ Указанная статья опубликована в 2014 году и может не отражать важных изменений, которые произошли с тех пор. — *Примеч. ред.*

² Сокр. Desktop Management Interface (интерфейс управления рабочим столом). — *Примеч. пер.*

стандарт SMBIOS¹. В этой классификации существует более 50 типов устройств: например, тип 4 относится к характеристикам процессора, а тип 43 — к устройствам TPM. Полную информацию об этих типах можно найти в официальной спецификации SMBIOS (dmtf.org/standards/smbios).

Например, чтобы просмотреть подробную информацию о системной памяти, можно запустить команду `dmidecode` с ключом `-t (--type)` и кодом 17, который соответствует устройствам памяти в SMBIOS:

```
packt@neptune:~$ sudo dmidecode -t 17
# dmidecode 3.5
SMBIOS 3.5 present.

Handle 0x000D, DMI type 17, 92 bytes
Memory Device
    Array Handle: 0x000C
    Error Information Handle: Not Provided
    Total Width: 64 bits
    Data Width: 64 bits
    Size: 48 GB
    Form Factor: DIMM
    Set: None
    Locator: Controller0-ChannelA-DIMM0
    Bank Locator: BANK 0
    Type: DDR5
    Type Detail: Synchronous
    Speed: 4800 MT/s
    Manufacturer: Kingston
    Serial Number: 2E069D27
    Asset Tag: 9876543210
    Part Number: KF560C32-48
    Rank: 2
    Configured Memory Speed: 4800 MT/s
    Minimum Voltage: 1.1 V
    Maximum Voltage: 1.35 V
    Configured Voltage: 1.1 V
    Memory Technology: DRAM
    Memory Operating Mode Capability: Volatile memory
    Firmware Version: Not Specified
    Module Manufacturer ID: Bank 2, Hex 0x98
    Module Product ID: Unknown
    Memory Subsystem Controller Manufacturer ID: Unknown
    Memory Subsystem Controller Product ID: Unknown
    Non-Volatile Size: None
    Volatile Size: 48 GB
    Cache Size: None
    Logical Size: None
```

Чтобы получить аналогичную информацию о других аппаратных компонентах, вызывайте `dmidecode` с соответствующими кодами.

¹ Сокр. System Management BIOS (BIOS управления системой). — *Примеч. пер.*

При устранении неполадок нередко помогают еще три команды:

lspci	Сведения о шинах PCI и устройствах
lsblk	Сведения о блочных устройствах (в частности, дисках и их разделах)
lscpu	Сведения о центральном процессоре

Кроме того, когда нужно диагностировать проблемы с оборудованием, нелишним будет взглянуть на журналы ядра — для этого служит команда `dmesg`. Ее можно запустить в виде `dmesg | more`, чтобы просматривать вывод было удобнее.

Как вы убедились в этом разделе, искать и устранять аппаратные неполадки — не менее важная и сложная задача, чем борьба со всеми остальными видами неполадок. Решать проблемы оборудования — неотъемлемая часть работы любого системного администратора, которому приходится постоянно проверять аппаратные компоненты, заменять неисправные комплектующие на новые и обеспечивать, чтобы они работали без накладок.

Итоги

В этой главе мы рассказали о том, как важно планировать аварийное восстановление, поддерживать стратегии резервного копирования и восстановления, а также о том, как устранять различные системные неполадки. Каждый системный администратор должен уметь применить свои знания на практике, если случится авария. Различные типы сбоев в конечном итоге затронут эксплуатационные серверы, поэтому нужно как можно скорее решать проблемы, чтобы свести к минимуму время простоя и не потерять данные.

Эта глава стала кульминацией части «Расширенные возможности администрирования Linux». В следующей части мы познакомим вас с **администрированием серверов**, уделив особое внимание управлению виртуальными машинами KVM, контейнерам Docker и различным типам конфигурации серверов Linux.

Вопросы

Прежде чем перейти к разделу о серверах, предлагаем вам проверить свои знания по устранению неполадок:

1. Попробуйте составить черновой DRP для своей локальной сети или малого предприятия.
2. Создайте резервную копию всей своей системы, используя правило 321.
3. Узнайте, какие 10 процессов в вашей системе потребляют больше всего процессорного времени.
4. Узнайте, какие 10 процессов в вашей системе потребляют больше всего оперативной памяти.

Часть 3

СЕРВЕРНОЕ АДМИНИСТРИРОВАНИЕ

В этой, третьей, части вы узнаете о продвинутых задачах администрирования серверов Linux. Вам предстоит настраивать различные типы серверов, а также управлять виртуальными машинами и контейнерами Docker.

Эта часть состоит из трех глав:

- *Глава 11. Как работать с виртуальными машинами*
- *Глава 12. Как управлять контейнерами с помощью Docker*
- *Глава 13. Как настраивать серверы Linux*

11

Как работать с виртуальными машинами

В этой главе вы познакомитесь с **виртуальными машинами** в Linux. Первым делом вы узнаете, как работает виртуализация, а также как создавать и использовать виртуальные машины. Мы рассмотрим KVM — одну из самых популярных технологий виртуализации и гипервизоров в Linux. Темы этой главы откроют перспективы будущего развития Linux, потому что именно эта система лежит в основе всех современных облачных технологий. Если вы хотите оставаться в курсе событий в этой динамично развивающейся области, эта глава станет для вас важной отправной точкой.

Мы рассмотрим такие темы:

- Введение в виртуализацию в Linux.
- Как устроен гипервизор KVM в Linux.
- Основные команды KVM.
- Расширенное управление KVM.
- Как оснащать виртуальные машины с помощью cloud-init.
- Аутентификация SSH с открытым ключом.

Технические требования

В этой главе нет никаких особых технических требований; достаточно иметь действующую систему Linux¹. В наших примерах мы в основном будем использовать Debian GNU/Linux 12, но также рассмотрим, как установить KVM в Fedora и openSUSE.

¹ Для упражнений с виртуальными машинами лучше, чтобы базовая ОС Linux (например, Debian) сама была не виртуальной, а запускалась непосредственно на физическом оборудовании, иначе можно столкнуться с проблемами вложенной виртуализации. — *Примеч. ред.*

Введение в виртуализацию в Linux

Виртуализация позволяет эффективнее использовать оборудование. По сути это слой абстракции поверх ресурсов компьютера. В этом разделе вы узнаете о том, какие бывают виртуальные машины, как они работают в Linux, а также как их развертывать и как ими управлять.

Как виртуализация помогает эффективнее использовать ресурсы

Уровень абстракции, на котором работает виртуализация, — это программный слой, благодаря которому удастся эффективнее использовать компоненты компьютера. Это, в свою очередь, позволяет оптимальнее использовать возможности и ресурсы физического оборудования.

Прежде чем углубиться в тему виртуализации, приведем пример. В нашей тестовой лаборатории есть несколько ноутбуков и настольных компьютеров малого форм-фактора (Intel NUC), которые мы используем в качестве серверов. У каждой из этих систем более чем достаточно ресурсов для целевых служб. Например, наши наименее производительные системы — это Intel NUC 5-го поколения с 4-ядерным процессором Intel i3 и 16 Гбайт памяти, а также Intel NUC 7-го поколения с 4-ядерным Intel Pentium и 12 Гбайт памяти. Оба компьютера располагают существенными мощностями, которые можно было бы эффективнее использовать с помощью виртуальных машин.

Чтобы запустить локальную веб-службу или любой другой сервер в нашей локальной сети, эти ресурсы можно легко распределить между различными виртуальными машинами. Например, на каждой физической системе можно разместить четыре разные виртуальные машины, каждой из которых достанется одно ядро процессора, не менее 2 Гбайт памяти и необходимый объем хранилища. В результате одна машина будет работать так, как будто их четыре¹. Это гораздо эффективнее, чем использовать для разных задач разные физические машины.

На следующей диаграмме мы условно сравниваем нагрузку на один компьютер с той же нагрузкой, разделенной между несколькими виртуальными машинами. Такое использование одних и тех же аппаратных ресурсов выглядит эффективнее:

¹ Здесь авторы описывают чрезмерно упрощенный подход, который не учитывает накладных расходов на виртуализацию. Та или иная часть аппаратных ресурсов уходит на работу базовой операционной системы и гипервизора, а также на обслуживание и координацию каждой виртуальной машины. Увеличивается время обращения к процессору и памяти, а кроме того, приходится особо управлять неразделяемыми ресурсами (ввод-вывод, сетевые интерфейсы и пр.). В реальной практике виртуализированные системы проектируют и профилируют с учетом накладных расходов и дополнительных рисков; важно убедиться, что в каждом случае виртуализация не создает проблем больше, чем решает. — *Примеч. ред.*



Рис. 11.1. Сравнение нагрузки на отдельный компьютер и на несколько виртуальных машин

Тем не менее, поскольку поверх базовой ОС будет работать гипервизор, нам придется выделить часть ресурсов для него и для самой ОС, поэтому потенциальное количество виртуальных машин будет меньше. Вот схема того, как они работают под управлением базовой ОС:

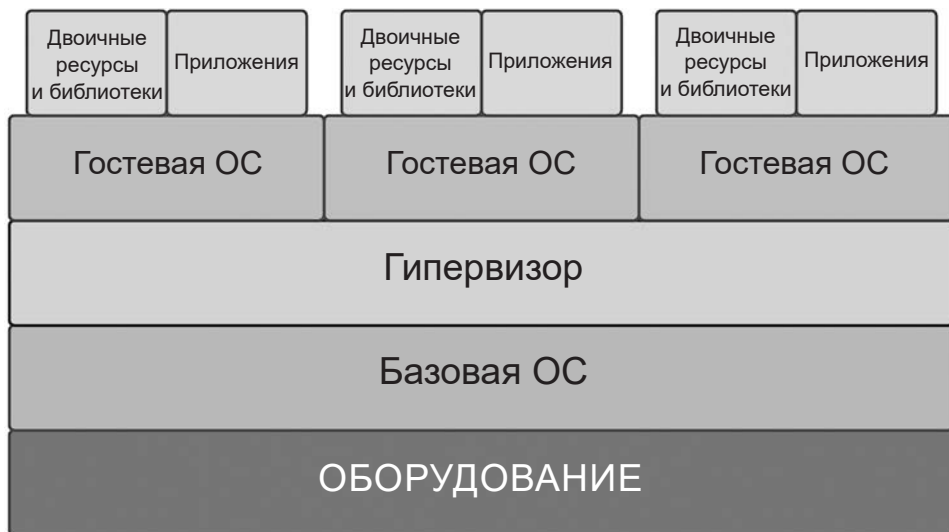


Рис. 11.2. Как виртуализация работает поверх базовой операционной системы

На предыдущей схеме показана виртуализация на основе базовой операционной системы. Как мы увидим в следующих разделах, это не единственный тип виртуализации.

Важно отметить, что эффективность связана не только с тем, как расходуются аппаратные ресурсы. Оптимизация использования оборудования в центрах обработки данных помогает повысить энергоэффективность и сократить углеродный след. В этом отношении виртуализация уже много десятилетий существенно влияет на то, как меняются модели использования серверов в центрах обработки данных. В целом виртуализация и контейнеризация играют важную роль в борьбе с изменением климата.

В следующих разделах мы представим вам краткий обзор гипервизоров и виртуальных машин.

Что такое гипервизоры

Программный уровень, на котором основана виртуализация, называется **гипервизором**. Физические ресурсы базового компьютера разделяются и используются так, будто они принадлежат отдельным виртуальным машинам. При этом ограничения физического оборудования преодолеваются за счет программной **эмуляции**. Это дает массу преимуществ, позволяя эффективнее использовать аппаратное обеспечение.

Эмуляция — это, по сути, эффект имитации, когда программное обеспечение воспроизводит (или имитирует) функции другой системы. В нашем случае гипервизор (программный слой виртуализации) имитирует нижележащее аппаратное обеспечение так, чтобы оно вело себя подобно настоящему оборудованию отдельной системы.

Гипервизоры могут использоваться как поверх существующей ОС (*тип 2*), так и непосредственно на физическом оборудовании (*тип 1*). Для каждого из этих типов существуют различные промышленные решения. Например, в Linux применяются такие продукты:

- Гипервизоры типа 2 (которые работают поверх базовой ОС) — Oracle VirtualBox и VMware Workstation/Fusion.
- Гипервизоры типа 1 (которые работают непосредственно поверх физического оборудования) — Xen Project и VMware ESXi.
- KVM в основном классифицируется как гипервизор аппаратного уровня (тип 1), однако он располагает полноценной собственной операционной системой, поэтому одновременно считается гипервизором поверх ОС (тип 2).

В этой главе мы будем использовать в качестве гипервизора исключительно KVM.

Как устроен гипервизор KVM в Linux

Виртуальная машина похожа на отдельный компьютер. Это программный эмулятор, у которого есть доступ к ресурсам базового компьютера — процессору, оперативной памяти, хранилищу, сетевому интерфейсу (или нескольким ин-

терфейсам) и портам. Кроме того, это виртуальная среда, которая способна выполнять те же функции, что и физический компьютер.

Ресурсами каждой виртуальной машины управляет гипервизор. Он может перераспределять ресурсы между существующими машинами или создавать новые виртуальные машины, которые изолированы друг от друга и от базового компьютера. Поскольку на одном компьютере может существовать несколько виртуальных машин, в каждой из них может быть своя гостевая ОС. Например, если вы работаете под Windows или macOS и хотите попробовать Linux, можно для начала установить интересующий вас дистрибутив в виртуальной машине. Операционная система, которая установлена в виртуальной машине, ведет себя в целом так же, как на физическом оборудовании. Разные гипервизоры предлагают разный пользовательский интерфейс и могут различаться другими качествами — например, степенью оптимизации ресурсов и временем отклика. Мы сами предпочитаем запускать виртуальные машины из KVM — в основном из-за богатого **интерфейса командной строки**. Однако у разных пользователей могут быть разные сценарии использования.

Особенности KVM

Для этой главы мы выбрали гипервизор KVM. В качестве бонуса, если вы используете среду рабочего стола GNOME, у вас будет доступ к GNOME Boxes. Поскольку и KVM, и GNOME Boxes доступны непосредственно из репозитория Linux, нам кажется, что они лучше всего подходят для новичков в Linux. Оба этих продукта основываются на `libvirt` и `qemu` (о которых мы расскажем в следующем разделе), и поэтому мы считаем их одним и тем же гипервизором и называем общим именем KVM.

В главе 1 «Как установить Linux» вы впервые познакомились с тем, как наладить виртуальную машину Linux под управлением гипервизора. Мы демонстрировали соответствующие упражнения на примере VMware и VirtualBox и считаем, что этого вводного материала должно хватить для любого пользователя — от новичка до профессионала. На наш взгляд, VirtualBox — неплохой гипервизор с богатыми возможностями, однако ему не хватает некоторого изящества KVM. В следующем разделе мы поговорим о KVM.

Как работать с гипервизором KVM

KVM¹ — это проект виртуализации с открытым исходным кодом, который доступен во всех основных дистрибутивах Linux. Это современный гипервизор со специальными модулями ядра, которые позволяют воспользоваться всеми современными преимуществами ядра Linux, включая поддержку памяти, планировщик, вложенную виртуализацию, сквозной доступ к GPU и т. д.

¹ Сокр. Kernel-based Virtual Machine (виртуальная машина на основе ядра). — *Примеч. пер.*

KVM в подробностях: QEMU и libvirt

KVM использует **QEMU** как программный эмулятор всех аппаратных компонентов и периферийных устройств. Основной инструмент управления и демон, который контролирует гипервизор, называется **libvirt**; он также служит в качестве **API** для KVM. Система мониторинга **virt-manager** позволяет управлять KVM с **libvirt** в средах с графическим интерфейсом (в первую очередь в GNOME), а утилита командной строки для **libvirt** называется **virsh**.

API **libvirt** предоставляет общую библиотеку для управления виртуальными машинами, которая позволяет их создавать, изменять и оснащать. Она работает в фоновом режиме как демон **libvirtd**, который управляет соединениями с гипервизором по запросу клиента.

QEMU — это одновременно эмулятор и виртуализатор. Когда он выступает в качестве эмулятора, он использует **динамическую двоичную трансляцию**: это значит, что на гостевых машинах могут функционировать различные типы операционных систем, даже если они предназначены для разных архитектур. Динамическая двоичная трансляция используется в **программной виртуализации**, которая эмулирует аппаратное обеспечение так, чтобы оно могло выполнять инструкции в виртуализированных средах. QEMU эмулирует центральный процессор машины с помощью **TSG** — специального механизма двоичной трансляции, который преобразует двоичный код для различных архитектур.

Когда QEMU применяется как виртуализатор, он задействует так называемую **аппаратную виртуализацию**, при которой двоичная трансляция не используется, потому что инструкции выполняются непосредственно на базовом процессоре. Различия между программной и аппаратной виртуализацией показаны на следующей схеме:



Рис. 11.3. Программная и аппаратная виртуализация

Как видно на этой диаграмме, при программной и аппаратной виртуализации инструкции проходят разные пути. При программной виртуализации с динамической двоичной трансляцией инструкции гостевой системы в непривилегированном режиме отправляются непосредственно на оборудование, а инструкции в привилегированном режиме сначала передаются гипервизору. При аппаратной виртуализации непривилегированные инструкции сначала направляются гипервизору, а затем — аппаратному обеспечению, в то время как привилегированные инструкции гостевой системы проходят тот же путь, что и при программной виртуализации. Это обеспечивает дополнительный уровень изоляции гостевой системы, благодаря чему улучшается производительность и сокращаются сложности.

В следующем разделе мы покажем, как установить и настроить QEMU на компьютере под управлением Debian 12. Мы считаем, что Debian — достаточно легковесный дистрибутив, который обеспечивает необходимую стабильность в качестве базовой системы для виртуализации. Некоторые команды можно повторить и на Ubuntu.

Как установить гипервизор в основных дистрибутивах Linux

Установить QEMU довольно просто: достаточно лишь запустить систему управления пакетами вашего дистрибутива, указав соответствующие имена пакетов. Вот как установить нужные пакеты в основных дистрибутивах Linux:

- **Debian/Ubuntu Linux¹**

```
sudo apt install qemu-kvm libvirt-clients libvirt-daemon-system  
bridge-utils virtinst libvirt-daemon virt-manager
```

- **Fedora Linux**

```
sudo dnf group install --with-optional virtualization
```

- **openSUSE Linux**

```
sudo zypper install -t pattern kvm_server kvm_tools  
sudo zypper install libvirt-daemon
```

¹ Чтобы установить минимальный комплект программного обеспечения, который позволяет запустить QEMU и KVM, можно запустить команду `sudo apt install --no-install-recommends qemu-system libvirt-clients libvirt-daemon-system` (см. wiki.debian.org/KVM). Обратитесь к официальным справочникам, чтобы уточнить современные рекомендуемые методы установки гипервизора и работы с ним: например, docs.fedoraproject.org/en-US/quick-docs/virtualization-getting-started для Fedora и docs.opensuse.org/documentation/leap/virtualization/html/book-virtualization для openSUSE. — *Примеч. ред.*

После того как вы установите все необходимые пакеты, можно включить и запустить демон `libvirtd` с помощью таких команд (они одинаковы во всех дистрибутивах Linux, которые упоминаются в этом разделе):

```
sudo systemctl enable libvirtd
sudo systemctl start libvirtd
```

Следующим этапом имеет смысл проверить совместимость вашей машины с требованиями KVM. Для этого запустите команду `virt-host-validate` от имени пользователя `root` или с помощью `sudo`:

```
sudo virt-host-validate
```

В зависимости от вашей системы эта команда может выдать несколько ошибок или предупреждений, которые касаются QEMU или LXC — технологии, которая позволяет запускать изолированные системы (подробнее см. в главе 12). В нашем случае вывод показывает одну ошибку совместимости с LXC:

```
packt@debian:~$ sudo virt-host-validate
QEMU: Checking for hardware virtualization : PASS
QEMU: Checking if device /dev/kvm exists : PASS
QEMU: Checking if device /dev/kvm is accessible : PASS
QEMU: Checking if device /dev/vhost-net exists : PASS
QEMU: Checking if device /dev/net/tun exists : PASS
QEMU: Checking for cgroup 'cpu' controller support : PASS
QEMU: Checking for cgroup 'cpuacct' controller support : PASS
QEMU: Checking for cgroup 'cpuset' controller support : PASS
QEMU: Checking for cgroup 'memory' controller support : PASS
QEMU: Checking for cgroup 'devices' controller support : PASS
QEMU: Checking for cgroup 'blkio' controller support : PASS
QEMU: Checking for device assignment IOMMU support : PASS
QEMU: Checking if IOMMU is enabled by kernel : PASS
QEMU: Checking for secure guest support : PASS
(Unknown if this platform has Secure Guest support)
LXC: Checking for Linux >= 2.6.26 : PASS
LXC: Checking for namespace ipc : PASS
LXC: Checking for namespace mnt : PASS
LXC: Checking for namespace pid : PASS
LXC: Checking for namespace uts : PASS
LXC: Checking for namespace net : PASS
LXC: Checking for namespace user : PASS
LXC: Checking for cgroup 'cpu' controller support : PASS
LXC: Checking for cgroup 'cpuacct' controller support : PASS
LXC: Checking for cgroup 'cpuset' controller support : PASS
LXC: Checking for cgroup 'memory' controller support : PASS
LXC: Checking for cgroup 'devices' controller support : PASS
LXC: Checking for cgroup 'freezer' controller support : FAIL
(Enable 'freezer' in kernel Kconfig file or mount/enable cgroup
controller in your system)
LXC: Checking for cgroup 'blkio' controller support : PASS
LXC: Checking if device /sys/fs/fuse/connections exists : PASS
```

Однако эта ошибка не мешает работать с `libvirt` и `QEMU`, поэтому здесь мы не будем ее устранять.

Убедившись, что нет критических проблем совместимости с `QEMU`, можно начать создавать нашу первую виртуальную машину с помощью командной строки. Для этого мы будем запускать команды, специфичные для `KVM`.

Как работать с основными командами KVM

Чтобы работать с `KVM`, в первую очередь понадобятся команды, с помощью которых можно создать виртуальную машину, а также команды, которые позволяют запустить, остановить, приостановить или удалить уже существующую машину. Мы рассмотрим их в следующих разделах.

Как создать виртуальную машину с помощью командной строки

Перед тем как создавать нашу первую виртуальную машину с помощью `libvirt`, давайте проверим, был ли создан сетевой мост по умолчанию:

```
sudo virsh net-list
```

Эта команда показывает, существует ли конфигурация моста по умолчанию и запущена ли она. В нашем случае мост не запущен, поэтому настроим его самостоятельно с помощью такой команды:

```
sudo virsh net-start default
```

Сам по себе сетевой мост не настроен на автоматический запуск, но это можно исправить такой командой:

```
sudo virsh net-autostart default
```

Теперь сетевая конфигурация выглядит так:

```
packt@debian:~$ sudo virsh net-list
  Name      State    Autostart  Persistent
-----
 default    active   yes        yes
```

Теперь, когда сетевой мост по умолчанию включен и запускается автоматически, можно создать нашу первую виртуальную машину. Для этого выполните следующие шаги:

1. Сначала нам понадобится файл образа операционной системы, которая будет установлена в виртуальной машине. В нашем примере мы используем серверную версию `Ubuntu 24.04.2 LTS`, образ `ISO` которой можно загрузить с помощью такой команды:

```
wget https://releases.ubuntu.com/noble/ubuntu-24.04.2-live-server-amd64.iso
```

2. Файл ISO будет загружен в ваш текущий рабочий каталог, но чтобы гипервизор мог обращаться к нему из стандартного расположения, файл следует переместить в каталог `/var/lib/libvirt/images`.
3. После того как образ Ubuntu загружен, мы создадим первую виртуальную машину в нашей системе с помощью команды `virt-install`. У нашей виртуальной машины будет один **виртуальный процессор**, 2 Гбайт оперативной памяти и 20 Гбайт хранилища.

Чтобы создать виртуальную машину, запустите такую команду с правами суперпользователя:

```
sudo virt-install \
  --virt-type=kvm \           # Тип новой виртуальной машины
  --name ubuntu-vm \         # Имя машины
  --vcpus=2 \                 # Количество виртуальных процессоров
  --memory=2048 \            # Объем памяти
  --osinfo ubuntu24.04 \      # Гостевая ОС
  --cdrom=/var/lib/libvirt/images/ubuntu-24.04.2-live-server-amd64.iso \
                              # Расположение установочного файла ISO
  --network=default \        # Настройки сети по умолчанию
  --disk size=20              # Объем диска
```

Эта команда запустит новое окно `virt-viewer`, в котором отобразится процесс установки ОС. Если добавить в команду аргумент `--graphics=vnc`, то виртуальная машина будет снабжена графической консолью, к которой можно будет обратиться по протоколу VNC.

Системному администратору недостаточно просто знать, как создать виртуальную машину. Поэтому в следующем разделе мы рассмотрим основные инструменты, с помощью которых ею можно управлять.

Как управлять виртуальными машинами

Основные операции с виртуальными машинами можно выполнять с помощью команды `virsh` в командной строке или Менеджера виртуальных машин (Virtual Machine Manager) в графическом интерфейсе. Далее мы рассмотрим работу в командной строке; все соответствующие команды нужно запускать от имени суперпользователя (например, с помощью `sudo` или из-под учетной записи `root`).

Чтобы перечислить запущенные гостевые виртуальные машины, используйте команду `virsh list`:

```
packt@debian:~$ sudo virsh list
Id   Name               State
-----
```

Вот основные подкоманды `virsh`, с помощью которых можно управлять виртуальными машинами. Все они запускаются в формате `sudo virsh подкоманда имя_машины`; например, чтобы перезагрузить виртуальную машину, запустите `sudo virsh reboot ubuntu-vm`.

Операция	Подкоманда
Принудительно остановить VM	destroy
Перезагрузить VM	reboot
Приостановить VM	suspend
Запустить остановленную VM	start
Возобновить работу приостановленной VM	resume
Полностью удалить VM	undefine

Чтобы узнать о том, какие еще операции можно выполнить с помощью `virsh`, обратитесь к соответствующей справочной странице:

```
man virsh
```

Инструменты командной строки достаточно мощны для того, чтобы полноценно работать с виртуальными машинами. Если учесть, что системные администраторы почти всегда используют командную строку, а не графический интерфейс, очень важно овладеть этими инструментами.

В следующем разделе мы поговорим о некоторых продвинутых приемах управления KVM.

Расширенное управление KVM

Использование KVM не ограничивается тем, чтобы просто создавать виртуальные машины, запускать и останавливать их. Работать с виртуальными машинами можно гораздо разнообразнее: например, автоматически устанавливать их, управлять хранилищем и ресурсами, а также оркестровать виртуальные машины. Некоторые из этих тем выходят за рамки текущей книги, но мы все равно рассмотрим несколько продвинутых приемов управления виртуальными машинами в системах на базе Linux.

До сих пор у нас была только одна виртуальная машина, но для упражнений из этого раздела мы создадим еще две машины под управлением той же Ubuntu 24.04 и назовем их `ubuntu-vm2` и `ubuntu-vm3`. Чтобы создать первую из них, запустите такую команду:

```
sudo virt-install --virt-type=kvm --name ubuntu-vm2 --vcpus=2 --memory=2048  
--osinfo ubuntu24.04 --cdrom=/var/lib/libvirt/images/ubuntu-24.04.2-live-server-  
amd64.iso --network=default --disk size=20 --noautoconsole
```

Оставшаяся виртуальная машина создается аналогично, только в качестве имени нужно указать `ubuntu-vm3`.

Теперь в нашей системе запущены три виртуальные машины, и ими можно управлять. В следующем разделе мы покажем, как узнать IP виртуальной машины и как к ней подключиться.

Как подключиться к виртуальной машине

Чаще всего имеет смысл подключаться к действующей виртуальной машине с помощью терминала, а не использовать встроенную консоль, которую предоставляет система управления виртуальными машинами. Чтобы подключиться, нужно знать IP-адрес машины. Для начала посмотрим, что покажет простая команда `ip neigh`:

```
packt@debian:~$ ip neigh
192.168.122.141 dev virbr0 lladdr 52:54:00:0c:b0:53 REACHABLE
192.168.122.119 dev virbr0 lladdr 52:54:00:fa:6d:d9 REACHABLE
192.168.50.1 dev ens33 lladdr 3c:7c:3f:e2:d6:d8 REACHABLE
192.168.50.19 dev ens33 lladdr 9c:6b:00:02:11:f9 REACHABLE
192.168.122.48 dev virbr0 lladdr 52:54:00:0a:2b:26 REACHABLE
fe80::6d37:c160:aaa:a51f dev ens33 lladdr 9c:6b:00:02:11:f9 STALE
fe80::3e7c:3fff:fee2:d6d8 dev ens33 lladdr 3c:7c:3f:e2:d6:d8 router STALE
```

Здесь перечислены все IP-адреса в нашей локальной сети, причем три из них принадлежат виртуальной сети по умолчанию, которую установил KVM (`virbr0`). Но какой адрес относится к какой виртуальной машине? Чтобы узнать подробности, сначала запросим список всех виртуальных машин:

```
packt@debian:~$ sudo virsh list --all
```

```
[sudo] password for rostislav:
```

Id	Name	State
1	ubuntu-vm1	running
2	ubuntu-vm2	running
3	ubuntu-vm3	running

Здесь показаны имена виртуальных машин, а чтобы увидеть, какой IP-адрес связан с каждой из них, можно использовать подкоманду `domifaddr`:

```
packt@debian:~$ sudo virsh domifaddr ubuntu-vm1
```

Name	MAC address	Protocol	Address
vnet6	52:54:00:0c:b0:53	ipv4	192.168.122.141/24

```
packt@debian:~$ sudo virsh domifaddr ubuntu-vm2
```

Name	MAC address	Protocol	Address
vnet7	52:54:00:0a:2b:26	ipv4	192.168.122.48/24

```
packt@debian:~$ sudo virsh domifaddr ubuntu-vm3
```

Name	MAC address	Protocol	Address
vnet8	52:54:00:fa:6d:d9	ipv4	192.168.122.119/24

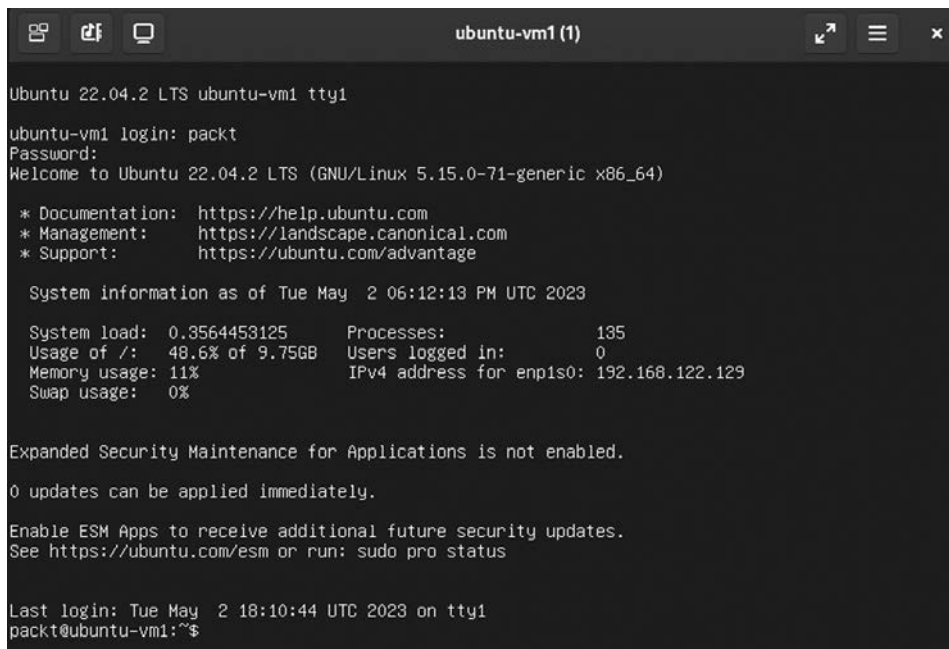
Теперь, когда мы знаем IP-адреса всех наших виртуальных машин, мы можем подключиться к любой из них с помощью SSH. (Подробнее о том, как установить и настроить SSH, см. в главе 13.) Учитывая, что и в базовой системе, и в виртуальных машинах уже установлен `openSSH`, к `ubuntu-vm1` проще всего будет подключиться так:

```
packt@debian:~$ ssh rostislav@192.168.122.141
The authenticity of host '192.168.122.141 (192.168.122.141)' can't be established.
ED25519 key fingerprint is SHA256:Kiz6p9X724anuZIRsSmpFq08KIHPNUP9LNGCI/OIpZQ.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.122.141' (ED25519) to the list of known hosts.
packt@192.168.122.141's password:
Welcome to Ubuntu 24.04.2 LTS (GNU/Linux 6.8.0-56-generic x86_64)
```

Другой способ подключиться к виртуальной машине — использовать команду `virt-viewer`:

```
virt-viewer --connect qemu:///system ubuntu-vm1
```

Эта команда откроет новое окно консоли с помощью утилиты `virt-viewer` и подключится к виртуальной машине `ubuntu-vm1`, не используя SSH:



```
ubuntu-vm1 (1)
Ubuntu 22.04.2 LTS ubuntu-vm1 tty1
ubuntu-vm1 login: packt
Password:
Welcome to Ubuntu 22.04.2 LTS (GNU/Linux 5.15.0-71-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

System information as of Tue May  2 06:12:13 PM UTC 2023

System load:  0.3564453125   Processes:            135
Usage of /:   48.6% of 9.75GB Users logged in:          0
Memory usage: 11%          IPv4 address for enp1s0: 192.168.122.129
Swap usage:   0%

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

Last login: Tue May  2 18:10:44 UTC 2023 on tty1
packt@ubuntu-vm1:~$
```

Рис. 11.4. Подключение к виртуальной машине с помощью `virt-viewer`

Соединение остается активным в том терминале, где вы инициировали команду. Таким образом, если вы нажмете **Ctrl + C**, то соединение прервется, а окно консоли закроется. Обратите внимание, что соединение разорвется, но виртуальная машина будет по-прежнему запущена.

Мы рассмотрели, как с помощью командной строки создавать виртуальные машины, управлять ими и подключаться к ним. Однако ничто не мешает также использовать для этого инструменты графического интерфейса. Все современные дистрибутивы Linux, в которых используется среда рабочего стола GNOME, предлагают как минимум два полезных инструмента: Менеджер виртуальных машин и GNOME Boxes. Первый из них — это просто графический интерфейс для `libvirt`, а второй — новый и простой способ оснащать виртуальные машины так, чтобы их можно было непосредственно использовать внутри GNOME; этот механизм опирается на `QEMU/KVM`. Мы рекомендуем вам самостоятельно познакомиться с этими инструментами, потому что они довольно понятны и просты в использовании. Теперь вы можете создавать новые виртуальные машины с помощью Менеджера виртуальных машин. В следующем разделе мы покажем, как клонировать виртуальные машины.

Как клонировать виртуальные машины

Мы уже создали три разные виртуальные машины в нашей базовой системе. Однако бывают случаи, когда нужно клонировать существующую машину, а не создавать новую.

Прежде чем клонировать машину, ее нужно остановить (`shutdown`) или поставить на паузу (`suspend`). Давайте остановим одну из наших виртуальных машин:

```
sudo virsh shutdown ubuntu-vm1
```

Теперь, чтобы клонировать ее, мы воспользуемся командой `virt-clone`. Допустим, назовем клон `ubuntu-vm1-clone1`:

```
packt@debian:~$ sudo virt-clone --original ubuntu-vm1 --name ubuntu-vm1-clone1
--auto-clone
Allocating 'ubuntu-vm1-clone1.qcow2' | 3.5 GB 00:00:04 ...
Clone 'ubuntu-vm1-clone1' created successfully.
```

```
packt@debian:~$ sudo virsh list --all
```

Id	Name	State
2	ubuntu-vm2	running
3	ubuntu-vm3	running
-	ubuntu-vm1	shut off
-	ubuntu-vm1-clone1	shut off

Теперь, когда клон готов, его можно запустить с помощью команды `virsh start`. Когда клонируется виртуальная машина, клон сохраняет *всю* конфигурацию исходной машины — в том числе количество виртуальных процессоров, объем оперативной памяти, конфигурацию сетевого моста и даже MAC-адрес и IP-адрес. Это может стать настоящей проблемой, которую понадобится решать.

Одно из решений состоит в том, чтобы напрямую подключиться к консоли виртуальной машины (не через SSH) и запустить команду `ip addr show` — в результате клиент DHCP автоматически назначит узлу IP-адрес. В следующем разделе мы покажем вам еще один практичный способ клонирования — с помощью шаблонов виртуальных машин.

Как работать с шаблонами виртуальных машин

Еще один способ преодолеть проблему, описанную в предыдущем разделе, — создать шаблон виртуальной машины перед тем, как ее клонировать. Благодаря шаблону можно обеспечить, чтобы не сохранились никакие элементы конфигурации, включая MAC- и IP-адреса, настройки пользователей или ключи SSH.

Шаблон можно создать так:

1. Мы будем использовать утилиту `virt-sysprep`. В Debian 12 ее можно установить в составе пакета `libguestfs-tools`:

```
sudo apt install libguestfs-tools
```

2. После того как утилита установлена, мы используем ее, чтобы создать шаблон. Но сначала давайте создадим еще одну новую виртуальную машину под управлением Ubuntu, которая будет выступать в качестве шаблона:

```
sudo virt-install --virt-type=kvm --name ubuntu-template --vcpus=2  
--memory=2048 --osinfo ubuntu24.04 --cdrom=/var/lib/libvirt/images/ubuntu-  
24.04.2-live-server-amd64.iso --network=default --disk size=20  
--noautoconsole
```

3. Установив гостевую систему, убедитесь, что она сама и все ее пакеты находятся в актуальном состоянии.
4. Выключите виртуальную машину перед тем, как превращать ее в шаблон. В качестве меры предосторожности можно сначала сохранить копию виртуальной машины под другим именем, например:

```
sudo cp /var/lib/libvirt/images/ubuntu-template.qcow2 /var/lib/libvirt/  
images/ubuntu-back-template.qcow2
```

5. Теперь подготовьте шаблон для клонирования с помощью утилиты `virt-sysprep`, которая «обнуляет» всю системную конфигурацию и пользовательские настройки. Вот выдержка из вывода соответствующей команды:

```

rostislav@debian:~$ sudo virt-sysprep -d ubuntu-template
[ 0.0] Examining the guest ...
[ 6.0] Performing "abrt-data" ...
[ 6.0] Performing "backup-files" ...
[ 6.3] Performing "bash-history" ...
[ 6.3] Performing "blkid-tab" ...
... и т. д. ...
[ 7.0] Setting a random seed
[ 7.0] Setting the machine ID in /etc/machine-id
[ 7.1] Performing "lvm-uuids" ...

```

6. Теперь, когда шаблон готов, можно действовать по-разному:

- Разрегистрируйте текущую виртуальную машину с помощью команды `virsh undefine`. Эта команда удаляет конфигурацию машины, но оставляет ее файл `qcow2`, с помощью которого можно создать новую виртуальную машину.
- Оставьте виртуальную машину (`ubuntu-template`) на месте и используйте ее в качестве шаблона для клонирования, как и предполагалось.

Выбор за вами, но мы склоняемся ко второму варианту, потому что он вам уже знаком и гораздо проще в использовании. Если вы воспользуетесь первым способом, то, прежде чем применять виртуальную машину, вам все равно придется настраивать ее характеристики — процессоры, оперативную память, сеть и т. д.

Теперь, когда вы умеете клонировать виртуальные машины и создавать шаблоны, давайте рассмотрим, как еще можно ими управлять. В следующем разделе мы продемонстрируем, как с помощью командной строки получать сведения о действующих виртуальных машинах.

Как получать сведения о ресурсах виртуальных машин и узлов

Когда вы работаете в командной строке, некоторая информация не так бросается в глаза, как в графическом интерфейсе. Если нужно проверить, достаточно ли на базовом компьютере ресурсов, чтобы создавать новые виртуальные машины, можно получить соответствующие сведения с помощью команды `virsh nodeinfo`:

```

alexandru@debian:~$ sudo virsh nodeinfo
CPU model:      x86_64
CPU(s):         16
CPU frequency:  3197 MHz
CPU socket(s):  1
Core(s) per socket: 8
Thread(s) per core: 2
NUMA cell(s):  1
Memory size:    48113628 KiB

```

Можете убедиться, что в нашем случае на базовом компьютере доступно 16 процессорных ядер и 48 Гбайт оперативной памяти, а это значит, что ресурсов хватит для того, чтобы создать еще несколько новых виртуальных машин. Вспомните, что для каждой из прежних машин мы выделяли по два виртуальных процессора и 2 Гбайт оперативной памяти. Поскольку сейчас у нас пять виртуальных машин, это значит, что мы используем 10 из 16 ядер и 10 из 48 Гбайт памяти:

```
alexandru@debian:~$ sudo virsh list --all
```

Id	Name	State
-	ubuntu-template	shut off
-	ubun u-vm1	shut off
-	ubuntu-vm1-clone1	shut off
-	ubuntu-vm2	shut off
-	ubuntu-vm3	shut off

Но что, если мы не знаем, сколько ресурсов используют существующие виртуальные машины? Есть команда `virsh dominfo`, которая может в этом помочь. Давайте посмотрим, какие ресурсы задействует одна из наших виртуальных машин, например `ubuntu-vm1`:

```
alexandru@debian:~$ sudo virsh dominfo ubuntu-vm1
```

```
Id: -
Name: ubuntu-vm1
UUID: 8b23c010-127b-4205-84f4-c77eefa76308
OS Type: hvm
State: shut off
CPU(s): 2
Max memory: 2097152 KiB
Used memory: 2097152 KiB
Persistent: yes
Autostart: disable
Managed save: no
Security model: apparmor
Security DOI: 0
```

Здесь видно, что наша виртуальная машина использует два виртуальных процессора и 2 Гбайт оперативной памяти. Аналогично можно проверить использование ресурсов для каждой машины. Помимо информации о процессоре и памяти, можно запросить сведения о виртуальных дисках — для этого служит команда `virt-df`:

```
alexandru@debian:~$ sudo virt-df -d ubuntu-vm1
```

Filesystem	1K-blocks	Used	Available	Use%
ubuntu-vm1:/dev/sda2	1790136	98240	1582636	6%
ubuntu-vm1:/dev/ubuntu-vg/ubuntu-lv	10218772	4492556	5185544	44%

```
alexandru@debian:~$ sudo virt-df -hd ubuntu-vm1
```

Filesystem	Size	Used	Available	Use%
ubuntu-vm1:/dev/sda2	1.7G	96M	1.5G	6%
ubuntu-vm1:/dev/ubuntu-vg/ubuntu-lv	9.7G	4.3G	4.9G	44%

Здесь ключ `-d` позволяет указать имя машины `libvirt`, а ключ `-h` выводит результаты в человекочитаемом формате. Команда `virt-df` аналогична команде `df` для базовой машины (см. главу 6).

Знать, как используются имеющиеся ресурсы, — это первый шаг к тому, чтобы управлять ими. В следующем разделе мы рассмотрим, как изменить объем ресурсов, которые потребляет виртуальная машина.

Как управлять использованием ресурсов виртуальной машины

Как вы наверняка уже убедились, очень полезно представлять себе, сколько ресурсов использует виртуальная машина, например, чтобы расширить их, если они подходят к концу. Посмотрим, как изменить количество виртуальных процессоров и объем RAM, которые использует существующая виртуальная машина. Например, давайте оставим нашей машине `ubuntu-vm1-clone1` один виртуальный процессор вместо двух и сократим память с 2 до 1 Гбайт. Для первой из этих операций понадобится команда `virsh setvcpus`:

```
alexandru@debian:~$ sudo virsh setvcpus ubuntu-vm1-clone1 1 --maximum --config
# Максимальное1 количество виртуальных процессоров
alexandru@debian:~$ sudo virsh setvcpus ubuntu-vm1-clone1 1 --config
# Реальное количество виртуальных процессоров
```

А чтобы изменить используемый и максимальный объем оперативной памяти, запустите соответственно команды `virsh setmem` и `virsh setmaxmem`:

```
alexandru@debian:~$ sudo virsh setmem ubuntu-vm1-clone1 1G --config
alexandru@debian:~$ sudo virsh setmaxmem ubuntu-vm1-clone1 1G --config2
```

Теперь можно снова проверить ресурсы, которые использует виртуальная машина `ubuntu-vm1-clone1`:

```
alexandru@debian:~$ sudo virsh dominfo ubuntu-vm1-clone1
Id:                -
Name:              ubuntu-vm1
UUID:              8b23c010-127b-4205-84f4-c77eefa76308
OS Type:           hvm
State:             shut off
CPU(s):            1
Max memory:        1048576 KiB
Used memory:       1048576 KiB
```

¹ «Максимальное» следует трактовать как то, сколько можно добавить на горячую работающую виртуальную машину. Подобное поведение позволяет добавлять и затем забирать предоставленные ресурсы для того, чтобы обеспечить работу операционной системы и прикладного программного обеспечения в случаях непредвиденного увеличения потребления ресурсов. — *Примеч. ред.*

² Ключ `--config` во всех перечисленных операциях приводит к тому, что изменения применяются после ближайшей перезагрузки виртуальной машины. — *Примеч. ред.*

```
Persistent:    yes
Autostart:     disable
Managed save: no
Security model: apparmor
Security DOI:  0
```

Как видите, ресурсы изменились в соответствии с новыми настройками. Теперь вы умеете управлять KVM, а это незаменимый навык для системного администратора Linux. В следующем разделе мы покажем, как автоматически инициализировать виртуальные машины KVM с помощью **cloud-init**.

Как управлять виртуальными машинами с помощью cloud-init

Когда у вас всего одна виртуальная машина, работать с ней может быть довольно просто. Но если нужно создать сотни виртуальных машин, вряд ли целесообразно заниматься этим вручную. Здесь приходят на помощь такие полезные инструменты, как **cloud-init** и **Ansible**. Об Ansible мы поговорим в главе 17, а в этом разделе подробно рассмотрим cloud-init. Он написан на Python, и его разработала компания Canonical в качестве инструмента, который позволяет конфигурировать экземпляры виртуальных машин. В настоящее время этот инструмент считается отраслевым стандартом для создания облачных образов. В следующем подразделе мы вкратце объясним, как он работает.

Как работает cloud-init

Работа cloud-init опирается на различные источники конфигурации, определенные этапы загрузки, форматы данных (пользователей и провайдера) и метаданные экземпляра. Разделение процесса загрузки на два этапа является специфическим для архитектуры cloud-init и позволяет применять различные настройки на том или ином этапе. С помощью этого инструмента можно управлять полностью рабочими экземплярами виртуальных машин, в которых настроены сеть, последовательность загрузки и локальные конфигурационные файлы.

cloud-init доступен в большинстве распространенных дистрибутивов Linux, таких как Ubuntu, Debian, RHEL, Fedora, SUSE и openSUSE. Мы установим и настроим cloud-init на одной из наших базовых систем под управлением Debian 12. Давайте посмотрим, как это сделать.

Как установить и настроить cloud-init

Несмотря на то что cloud-init предназначен в первую очередь для облачных развертываний, его можно применять и локально. С его помощью мы будем

развертывать виртуальные машины в своей локальной системе. Перед тем как использовать cloud-init, в системе уже должен быть установлен гипервизор, например KVM. В качестве гостевых образов cloud-init использует специальные облачные образы, которые доступны почти у всех поставщиков дистрибутивов Linux. Например, в этом разделе мы планируем работать с виртуальными машинами Ubuntu, поэтому нам понадобятся оптимизированные облачные образы с этим дистрибутивом, которые можно загрузить с сайта cloud-images.ubuntu.com. Ниже приведены необходимые действия, позволяющие развернуть виртуальную машину:

1. Сначала установите сам cloud-init, а также дополнительный пакет cloud-image-utils:

```
sudo apt install cloud-init cloud-image-utils
```

2. Создайте каталог для новых облачных образов и перейдите в него:

```
mkdir local-cloud-images && cd local-cloud-images
```

3. Загрузите нужный облачный образ:

```
wget https://cloud-images.ubuntu.com/noble/current/noble-server-cloudimg-amd64.img
```

4. Просмотрите подробные сведения об этом образе:

```
qemu-img info noble-server-cloudimg-amd64.img
```

5. Расширьте размер образа до 15 Гбайт:

```
qemu-img resize jammy-server-cloudimg-amd64.img 15G
```

6. Создайте новый образ в качестве базового образа .qcow2:

```
qemu-img create -F qcow2 -f qcow2 -b noble-server-cloudimg-amd64.img ubuntu.qcow2
```

7. Скопируйте образы в каталог libvirt:

```
sudo cp noble-server-cloudimg-amd64.img ubuntu.qcow2 /var/lib/libvirt/images/
```

Сейчас у нас есть образ, готовый к использованию, но нужно выполнить еще несколько шагов, чтобы настроить cloud-init. Мы создадим локальный файл для метаданных meta-data и еще один файл для пользовательских данных user-data. Это файлы в формате **YAML**, поэтому вам нужно будет использовать соответствующий синтаксис.

8. Создайте внутри /var/lib/libvirt/images/ новый каталог cloud-init, где будут храниться новые конфигурационные файлы. Это можно сделать такими командами (с правами суперпользователя):

```
mkdir cloud-init && cd cloud-init/ && touch user-data && touch meta-data && touch network-config
```

9. Отредактируйте файл user-data, как показано ниже.

На этом этапе нам понадобится пара ключей SSH, с помощью которых мы будем подключаться к будущей виртуальной машине. Поскольку мы еще не рассказали, как работать с ключами SSH, предоставим необходимую информацию в следующем разделе «Аутентификация SSH с открытым ключом». Прочтите его, а затем вернитесь к этому пункту и продолжайте редактировать файлы cloud-init.

Содержимое файла user-data должно выглядеть так:

```
#cloud-config

hostname: ubuntu-vm1
fqdn: ubuntu-vm1.localdomain
manage_etc_hosts: true

ssh_pwauth: false
disable_root: true

users:
- name: packt
  home: /home/packt
  shell: /bin/bash
  groups: sudo
  sudo: ["ALL=(ALL) NOPASSWD:ALL"]
  ssh_authorized_keys:
    - ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQGC7esAA44LcIT7kSU0L/O+BXDmgaGepc
      X7J5iWz0Z9hYMjJP+6BBZzBt8B7jhTtBZsNxHSv4nje9pCbN9bXJl0kYYMXLksDTjErXn59R/1E8H
      d8seCgle20hCjNcADQPMBOY9fLrJTPnAfIcJhCA6yETVTR9uiHn1e0WRZfadYzmf9K2buYzAxvR+
      3B2F/1qpb1zAOUWYX8sLSJ+J0ec9jVjGc35b30B4cydAQNVZbBHo2eIg4PAMdPwDXH6BFJBfHW8Yd
      GQxQMDlxrykeNoCK7Ep7qL7wrSo/RSS3hucloCo8sSj4PTosW1Rf4hhuJ5SbDe80+jLy/tUsj9J+U
      fAQxTcb19Bonhd+JaLPm3N0JFiK0kHrLKuYjrk6Pyx1GTVPQpSdvx+o0m1TBQ32VUfw+negXtNpFW
      OGmHNGJPyrBf6dmmYyn5bf6IcUL77Gt0esFEzv/JXYQ9Su3LWDMMLNptcvvXNM2gxpWE1DN3a1YBC
      0Jfg+SfGAPd5BaAjsA5k= packt@debian
```

10. Теперь можно продолжить создавать образ диска, который будет содержать конфигурационные файлы. Для этого запустите команду `cloud-localds` из пакета `cloud-image-utils`, который вы установили ранее:

```
cloud-localds -v ubuntu-provisioning.qcow2 user-data meta-data
```

11. Итак, мы создали новый образ диска на основе файла user-data. Подготовка закончена, и можно приступить к развертыванию. Разверните новую виртуальную машину с помощью такой команды:

```
packt@debian:~$ sudo virt-install --name vm01 --virt-type kvm --vcpus 1
--memory 2048 --disk path=/var/lib/libvirt/images/ubuntu.qcow2,device=disk
--disk path=/var/lib/libvirt/images/cloud-init/ubuntu-provisioning.
qcow2,device=cdrom --osinfo ubuntu24.04 --import --network network=default
--noautoconsole
```

```
Starting install...
Creating domain...
Domain creation completed.
```

```
| 0 B 00:00:00
```

12. Если вы сталкиваетесь с ошибками при инициализации сети, возможно, вам придется активировать сеть по умолчанию с помощью уже знакомых команд:

```
sudo virsh net-start default && sudo virsh net-autostart default
```

13. Вот мы и развернули новую виртуальную машину с помощью cloud-init. Чтобы проверить, что она работает, можно запустить команду `sudo virsh list` или использовать графический интерфейс Менеджера виртуальных машин:

```
packt@debian:~$ sudo virsh list
  Id   Name               State
  ----
  6    vm01                running
```

14. Убедившись, что виртуальная машина запущена, давайте узнаем ее IP-адрес и подключимся к ней с помощью SSH:

```
packt@debian:~$ sudo virsh domifaddr vm01
  Name      MAC address      Protocol  Address
  -----
  vnet4     52:54:00:81:0d:7c  ipv4     192.168.122.55/24

packt@debian:~$ ssh packt@192.168.122.55
Welcome to Ubuntu 24.04.2 LTS (GNU/Linux 6.8.0-55-generic x86_64)
```

Итак, мы успешно создали новую виртуальную машину с помощью cloud-init и подключились к ней по SSH. Однако здесь мы лишь поверхностно изучили возможности cloud-init, так что если вы хотите узнать больше об этом инструменте, обратитесь к официальной документации или другим источникам, которые перечислены в разделе «Дополнительная литература» в конце этой главы.

В следующем разделе мы поговорим о том, как настроить аутентификацию SSH с открытым ключом.

Аутентификация SSH с открытым ключом

Аутентификация с открытым ключом — это безопасный способ входа на виртуальный частный сервер или на облачный экземпляр с помощью SSH. Этот механизм обеспечивает большую безопасность и криптографическую стойкость, чем любой сложный пароль. Чтобы настроить аутентификацию с ключами SSH, нужно сгенерировать пару из двух ключей — закрытого и открытого. Закрытый ключ будет храниться на базовом локальном компьютере, а открытый будет использоваться в виртуальной машине. Ключи хранятся в подкаталоге `.ssh` в домашнем каталоге пользователя.

Чтобы сгенерировать новую пару ключей, можно воспользоваться командой `ssh-keygen`. Она может использоваться с разными аргументами, наиболее важные из которых — `-t` (алгоритм шифрования) и `-b` (разрядность ключа). Если

запустить `ssh-keygen` без аргументов, она будет использовать алгоритм шифрования RSA и 3072-битный ключ. Вот результат запуска этой команды:

```
packt@debian:~$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/packt/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/packt/.ssh/id_rsa
Your public key has been saved in /home/packt/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:NUJu96p5qsJjYhimGCNKENTxLJ8zldAeGaVoNzQDOzg packt@debian
The key's randomart image is:
+---[RSA 3072]-----+
|      .o*+.      |
|    ...*++      |
|    .E+=oBo+    |
|    . ooo++= o   |
|..   o oS   .   |
|*o.   =   .     |
|B* .   o   .    |
|= o =   o.      |
| . o o..+o      |
+----[SHA256]-----+
```

Как уже говорилось, оба ключа хранятся в каталоге `.ssh`. Закрытый ключ называется `id_rsa`, а открытый — `id_rsa.pub`. Чтобы настроить `cloud-init`, нам понадобится открытый ключ. Давайте убедимся, что ключи хранятся в нужном месте, и просмотрим открытый ключ:

```
packt@debian:~$ cd .ssh/

packt@debian:~/.ssh$ ls -la
total 16
drwx----- 2 packt packt 4096 Mar 26 23:04 .
drwx----- 16 packt packt 4096 Mar 26 22:52 ..
-rw----- 1 packt packt 2602 Mar 26 21:08 id_rsa
-rw-r--r-- 1 packt packt 570 Mar 26 21:08 id_rsa.pub

packt@debian:~/.ssh$ cat id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQGC7esAA44LcIT7kSUOL/O+BXDmgaGepcX7J5iWz0Z9h
YMJp+6BBZzBt8B7jhTtBZsNxHsv4nje9pCbN9bXJl0kYMYMLksDTjErXn59R/1E8Hd8seCgle20hCjN
cADQPMBQOY9fLrJTPnAfIcJhCA6yETVTR9uiHn1eOWRZfadYzmf9K2buYzAxvR+3B2F/lqpb1zAOUWYX
8sLSJ+J0ec9jVjGc35b30B4cydAQNVZbBH02eIg4PAMdPwDXH6BFJBfHW8YdGQxQMD1xrykeNoCK7Ep7
qL7WrSo/RSS3hucloCo8sSj4PTosW1Rf4hhuJSSbDe80+jLy/tUsj9J+UfAQxIcb19Bonhd+JaLPm3N0
JFik0kHrLKuYjrk6Pyx1GTVQPpSdvx+o0m1TBQ32VUfw+negXtNpfWOGmHNGJPyrBf6dmmYyn5bf6IcU
L77Gt0esFEzv/JXYQ9Su3LWDMMLnptcvvXNM2gxpWE1DN3a1YBC0JfG+SfGAPd5BaAjxA5k=
packt@debian
```

Однако если требуется с помощью этих ключей подключиться к облачному экземпляру виртуального частного сервера или к виртуальной машине, открытый ключ нужно безопасно скопировать в эту систему. Для этого мы восполь-

зуемся командой `ssh-copy-id`, при запуске которой понадобится указать имя пользователя и IP-адрес или имя узла для целевой машины. Например, если нужно скопировать открытый ключ SSH пользователя `packt` на виртуальную машину с IP-адресом `192.168.122.55`, используйте такую команду:

```
ssh-copy-id packt@192.168.122.48
```

В главе 13 мы подробнее расскажем о том, как установить и настроить сервер SSH, а приведенной здесь информации должно быть достаточно для того, чтобы подключиться к виртуальной машине, которую мы создали с помощью `cloud-init`.

Виртуализация — это важный раздел компьютерных технологий. Она помогает эффективно использовать огромную вычислительную мощность, которую предоставляют современные системы, и позволяет получить максимальную отдачу от инвестиций в аппаратные компоненты.

Итоги

В этой главе мы рассказали о том, как важна виртуализация в системах Linux. Мы показали, как создавать виртуальные машины и управлять ими с помощью KVM. Теперь вы умеете клонировать виртуальные машины, создавать их шаблоны и управлять их ресурсами. Вы знаете, как работает виртуализация и как установить гипервизор QEMU/KVM в Linux. Благодаря этим знаниям вы готовы уверенно работать с технологиями виртуализации в практической деятельности.

В следующей главе вы познакомитесь с контейнерными технологиями Docker.

Вопросы

Вот несколько вопросов об основных понятиях, которые рассматривались в этой главе:

1. Перечислите и охарактеризуйте типы гипервизоров.
2. В качестве практического упражнения установите гипервизоры в различных дистрибутивах Linux.
3. Проверьте, правильно ли работает ваш гипервизор.

Подсказка: используйте `virt-host-validate`.

4. Каковы основные различия между ведущими гипервизорами?

Совет: поэкспериментируйте, например, с KVM и VirtualBox и сравните их.

5. Как узнать IP-адреса виртуальных машин?

Подсказка: используйте команду `virsh domifaddr`.

Дополнительная литература

Чтобы получить дополнительную информацию о том, что рассматривалось в этой главе, можно обратиться к следующим книгам Packt:

- Ведран Дакич (Vedran Dakic), Хамбл Девасси Чираммал (Humble Devassy Chirammal), Прасад Мухедкар (Prasad Mukhedkar), Анил Веттатху (Anil Vettathu). «Mastering KVM Virtualization», Second Edition.
- Константин Иванов. «KVM Virtualization Cookbook».¹

Подробные сведения о внутреннем устройстве cloud-init можно найти на официальном сайте: cloudinit.readthedocs.io/en/latest/explanation.

¹ Некоторые книги и ресурсы в интернете, на которые ссылаются авторы, опубликованы много лет назад и могут не учитывать важных изменений и инноваций в индустрии, особенно в такой бурно развивающейся области, как виртуализация и контейнеризация. Хотя известны классические образцы компьютерной литературы, которые остаются востребованными и актуальными в течение долгих лет и даже десятилетий, профессиональному системному администратору важно также оставаться в курсе современных тенденций и следить за новейшими разработками. — *Примеч. ред.*

12

Как управлять контейнерами с помощью Docker

В этой главе вы узнаете о **Docker**: это один из самых известных инструментов, который позволяет создавать контейнеры и управлять ими.

Мы рассмотрим такие темы:

- Что такое контейнеры Linux.
- Основы работы с Docker.
- Как работать с Docker-файлами.
- Как разворачивать контейнерные приложения с помощью Docker.

Технические требования

В этой главе нет никаких особых технических требований; достаточно иметь действующую систему Linux. Для упражнений этой главы одинаково хорошо подходят как Ubuntu/Debian, так и Fedora/RHEL. В наших примерах мы в основном будем использовать Debian GNU/Linux 12, но там, где это необходимо, обсудим особенности Fedora Linux.

Что такое контейнеры Linux

Как мы продемонстрировали в предыдущей главе, существует два основных типа виртуализации: на основе **виртуальных машин** и на основе **контейнеров**. Ранее мы уже рассмотрели виртуальные машины, а теперь пришло время разобраться, что такое контейнеры. На самом элементарном уровне они подобны виртуальным машинам и похожим образом позволяют запускать изолированную среду, но на практике различий гораздо больше, чем сходства. Давайте подробнее сравним два этих понятия.

Контейнеры и виртуальные машины

Как вы уже знаете, виртуальная машина эмулирует аппаратное обеспечение и позволяет использовать один физический компьютер так, как будто это не-

сколько отдельных компьютеров. В отличие от этого контейнеры не имитируют физическое оборудование и ничего не эмулируют.

Контейнер предоставляет в распоряжение каждого из целевых приложений одно и то же базовое ядро операционной системы с общими библиотеками и двоичными файлами. Приложения находятся внутри контейнера и изолированы от остальной системы. Они также используют сетевой интерфейс совместно с базовой системой, чтобы обеспечить такое же подключение, как в виртуальных машинах.

Контейнеры запускаются поверх **контейнерного движка**, который обеспечивает виртуализацию на уровне ОС и позволяет развертывать и тестировать приложения, используя только необходимые библиотеки и зависимости. Таким образом, контейнеры заботятся о том, чтобы приложение могло работать на любой машине и вести себя так же, как задумал разработчик. Давайте наглядно сравним контейнеры с виртуальными машинами:

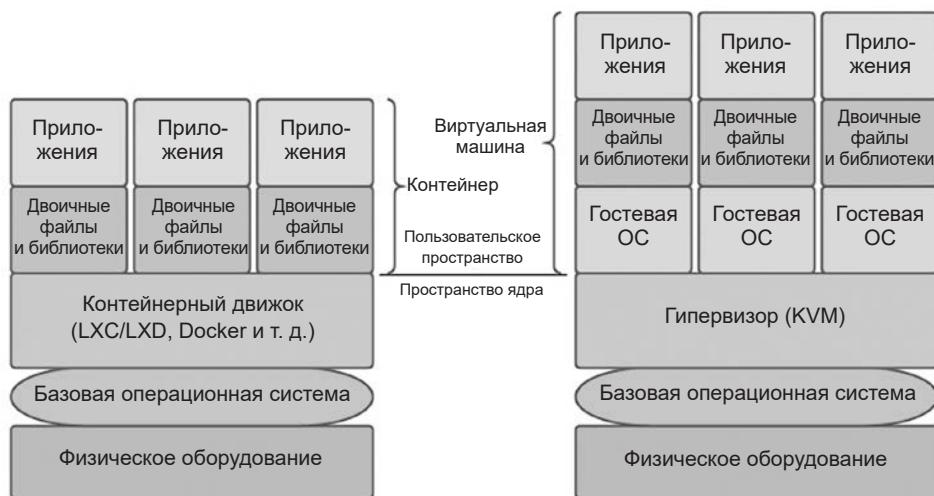


Рис. 12.1. Контейнеры и виртуальные машины (общая схема)

Как видите, контейнеры работают только в **пользовательском пространстве**, опираясь на общую архитектуру на уровне ОС.

С исторической точки зрения контейнеризация существует уже довольно долго; например, в Unix соответствующий инструмент **chroot** появился в конце 1970-х.

В Linux с конца 2000-х применялись контейнеры LXC, а в 2015 году появилась LXK — обновленная система управления контейнерами LXC. Наконец, с 2013 года развивается проект Docker.

В следующем разделе мы рассмотрим технологию, которая лежит в основе контейнеров.

Основы контейнерных технологий

Как уже упоминалось, одной из ранних форм контейнеров были LXC. Docker — это более новая технология, которая изменила всю индустрию и положила начало шумихе вокруг DevOps (подробнее об этом см. в главе 14). В отличие от гипервизоров, контейнеры не абстрагируют аппаратный уровень, а используют специфический интерфейс пользовательского пространства, который опирается на функциональность ядра, чтобы изолировать те или иные ресурсы. С помощью контейнеров можно воспроизвести стандартную систему Linux, не используя отдельное ядро, как в случае виртуальной машины.

Когда появились контейнеры LXC, одним из их преимуществ были богатые API для многих языков программирования, включая Python 3, Go, Ruby и Haskell. Поэтому, хотя LXC уже не так популярны, стоит иметь о них хотя бы общее представление. Мы не будем использовать LXC/LXD в примерах, но упоминаем о них для полноты картины. Наиболее свежая версия LXC на момент написания этой книги — 6.0, с поддержкой до июня 2029 года.

По словам разработчиков, LXC позволяют создать изолированную среду, максимально приближенную к стандартной установке Linux. Эта технология использует важные функции ядра, на которые опираются любые контейнерные решения в Linux: **пространства имен** ядра и **контрольные группы**. Кроме них, существуют механизмы chroot и профили безопасности для AppArmor и SELinux.

Давайте рассмотрим основные функции, на которые опираются контейнеры Linux.

Пространства имен Linux

Что такое **пространства имен** в Linux? В двух словах: это глобальные системные ресурсы ядра, отвечающие за изоляцию, которую обеспечивают контейнеры. Пространства имен оборачивают глобальные системные ресурсы в слой абстракции, благодаря чему прикладной процесс, который запускается внутри пространства имен, думает, что монопольно распоряжается тем или иным ресурсом. Пространство имен обеспечивает изоляцию на логическом уровне внутри ядра, а также предоставляет область видимости для каждого запущенного процесса.

Чтобы лучше понять, как работают пространства имен, представьте себе, как выглядят различные системные ресурсы и процессы с точки зрения произвольного пользователя Linux. Например, вам как пользователю видны глобальные системные ресурсы, запущенные процессы, другие пользователи и модули ядра.

Такая прозрачность может навредить, если мы хотим использовать контейнеры в качестве виртуализированной среды на уровне ОС. Поскольку контейнеры не могут обеспечить такой же уровень инкапсуляции и эмуляции, как виртуальные машины, контейнерный движок должен как-то преодолеть это ограничение, и низкоуровневые механизмы виртуализации среды в ядре представлены в виде пространств имен и контрольных групп.

В ядре Linux существует несколько типов пространств имен:

- **Файловая система (mnt)**. В пределах одного пространства имен видимость доступных точек монтирования файловой системы ограничивается так, что процессы из этого пространства обращаются к общему уникальному представлению файловой системы. У процессов могут быть собственные корневые файловые системы и различные частные или общие точки монтирования.
- **Разделение времени Unix (uts)** изолирует имя узла и доменное имя системы.
- **Межпроцессное взаимодействие (ipc)** позволяет процессам иметь собственную общую память IPC, очереди и семафоры.
- **Идентификация процессов (pid)** позволяет отображать идентификаторы PID так, чтобы процесс, который отображается на PID 1, порождал новое дерево с собственным корневым процессом. Процессы в пространстве имен PID видят только процессы в том же пространстве имен.
- **Сети (net)**. Абстракция на уровне сетевых протоколов. Процессы внутри сетевого пространства имен оперируют частным сетевым стеком с частными сетевыми интерфейсами, таблицами маршрутизации, сокетами и правилами iptables.
- **Пользователи (user)**. Позволяет транслировать UID и GID, в том числе отобразить непривилегированную учетную запись на корневой UID 0.
- **Контрольные группы (cgroup)**. Процесс в пространстве имен контрольной группы видит пути в файловой системе относительно корневого каталога пространства имен.

Доступные пространства имен можно просмотреть с помощью команды `lsns`, например:

```
alexandru@debian:~$ lsns
      NS TYPE      NPROCS  PID USER      COMMAND
4026531834 time        141   5097 alexandru /lib/systemd/systemd --user
4026531835 cgroup        141   5097 alexandru /lib/systemd/systemd --user
4026531836 pid         116   5097 alexandru /lib/systemd/systemd --user
4026531837 user         89   5097 alexandru /lib/systemd/systemd --user
4026531838 uts         141   5097 alexandru /lib/systemd/systemd --user
4026531839 ipc         115   5097 alexandru /lib/systemd/systemd --user
4026531840 net          115   5097 alexandru /lib/systemd/systemd --user
4026531841 mnt          113   5097 alexandru /lib/systemd/systemd --user
```

В следующем разделе мы рассмотрим контрольные группы — второй ключевой компонент контейнеров.

Контрольные группы Linux

Контрольные группы — это функции ядра, которые ограничивают распределение ресурсов между процессами и управляют этим распределением — в частности, тем, как используется память, процессор, ввод-вывод и сеть. Они предоставляют механизм, определяющий конкретные наборы задач (группы процессов), которые лимитируют, сколько ресурсов может использовать тот или иной процесс. Контрольные группы организованы в **иерархические** структуры: каждая дочерняя группа наследует атрибуты своей родительской группы, и в одной системе может одновременно существовать несколько иерархий контрольных групп.

Контрольные группы совместно с пространствами имен формируют изоляцию, на которую опираются контейнеры. Благодаря контрольным группам и пространствам имен ресурсы выделяются и управляются для каждого контейнера отдельно. Контейнеры потребляют гораздо меньше ресурсов, чем виртуальные машины, и функционируют как изолированные сущности.

Как уже говорилось, на практике используются контейнеры двух типов: LXC и Docker. LXC мы уже обсудили, так что в следующем разделе давайте разберемся, что такое Docker.

Что такое Docker

Docker, подобно LXC/LXD, основан на нескольких технологиях, главные из которых — пространства имен ядра и контрольные группы. Docker — это платформа, которая позволяет разрабатывать и распространять приложения; она обеспечивает базовую инфраструктуру для безопасной работы контейнеров.

Контейнеры Docker — это легковесные компоненты, которые запускаются непосредственно на ядре нижележащей системы, а платформа предлагает инструменты, с помощью которых можно создавать изолированные контейнерные приложения и управлять ими. Таким образом, контейнер — это атомарная единица разработки, тестирования и распространения приложений. Когда приложения готовы к эксплуатации и пригодны для развертывания, они могут поставляться в виде контейнеров или в виде оркестрованных служб. (Об оркестровке мы поговорим в главе 16 «Как развертывать приложения с помощью Kubernetes».)

Следующая схема демонстрирует, как устроена архитектура Docker:

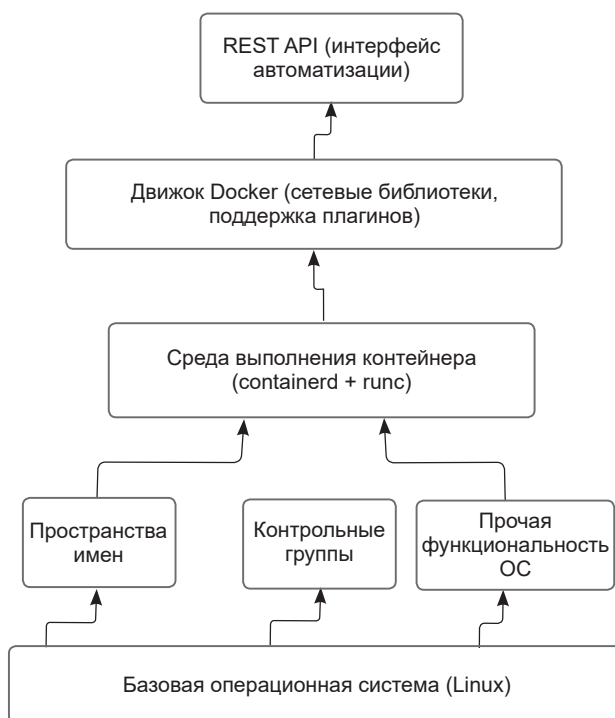


Рис. 12.2. Архитектура Docker

Поясним эту схему. Docker использует пространства имен и контрольные группы, доступные в ядре Linux, и состоит из двух основных компонентов: среды выполнения контейнеров и движка Docker, как показано на рис. 12.3.

Рассмотрим эти компоненты подробнее.

- **Среда выполнения контейнеров** складывается из двух частей:
 - `containerd` отвечает за то, как загружаются и запускаются образы Docker;
 - `runc` соответствует спецификациям OCI и управляет пространствами имен и контрольными группами для каждого контейнера;

OCI¹ — это проект в рамках Фонда Linux, который разрабатывает открытые отраслевые стандарты для контейнеров. Спецификация среды выполнения OCI определяет, как загрузить образ, распаковать его и запустить с помощью определенной структуры файловой системы. Компания Docker передала `runc` и `containerd` в фонд CNCF², чтобы больше организаций могли внести вклад в развитие обеих технологий.

¹ Сокр. Open Container Initiative (инициатива открытых контейнеров). — *Примеч. пер.*

² Сокр. Cloud Native Computing Foundation (фонд естественных облачных вычислений), входит в Фонд Linux. — *Примеч. пер.*

- В движок **Docker** входят демон `dockerd`, **API** и интерфейс командной строки.

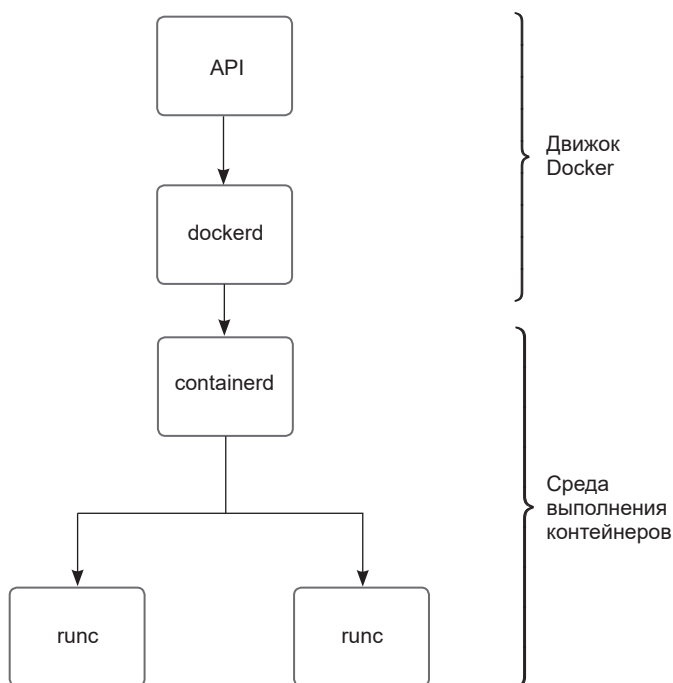


Рис. 12.3. Особенности архитектуры Docker

Помимо компонентов, которые перечислены в этом списке, в запуске и развертывании контейнеров Docker участвуют другие составляющие. Docker построен в соответствии с клиент-серверной архитектурой, и в рабочих процессах участвуют сервер Docker (на котором функционирует серверный демон), клиент и реестр. На сервере расположены образы и контейнеры, которые загружаются из реестра, а клиент предоставляет команды, с помощью которых можно управлять контейнерами.

Эти компоненты работают так:

- Демон `dockerd` прослушивает запросы API для управления службами и объектами (такими как образы, контейнеры, сети и тома).
- С помощью клиента пользователи могут взаимодействовать с демоном через API.
- В реестрах хранятся образы. В частности, Docker Hub — это публичный реестр, которым могут пользоваться все желающие. Кроме того, существуют частные реестры Docker.

Вот схема рабочих процессов Docker, на которой показаны клиентский компонент, API и демон:

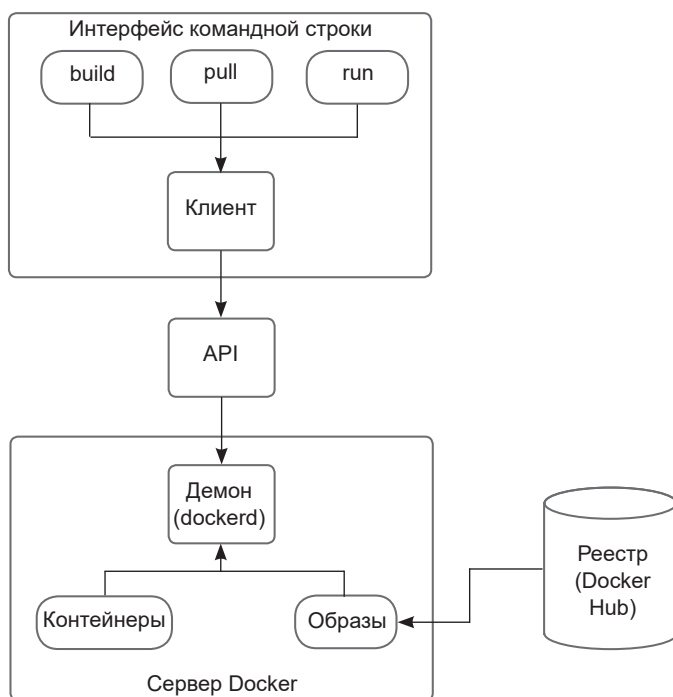


Рис. 12.4. Рабочие процессы Docker

Для новичка Docker может показаться сложным и даже пугающим. Множество компонентов, которые работают вместе, новые типологии и специфические рабочие процессы — все это действительно непросто. Этот раздел не претендовал на то, чтобы дать вам исчерпывающее представление о Docker; сейчас вы только в начале пути. Важно, чтобы у вас был прочный фундамент, на который сможет опираться дальнейший материал. Поэтому в следующем разделе вы узнаете, как установить и использовать Docker.

Основы работы с Docker

Для упражнений в этом разделе мы будем использовать Debian GNU/Linux 12, который установлен на виртуальной машине с двумя виртуальными процессорами и 2 Гбайт оперативной памяти. Но прежде чем устанавливать Docker, давайте немного подробнее остановимся на том, какие варианты Docker бывают и какой из них выбрать в нашем случае.

Какую версию Docker выбрать

Компания Docker, Inc, которая выпускает Docker, предлагает несколько продуктов на основе общей программной инфраструктуры. В последнее время в ее портфолио появились различные планы подписки на Docker, такие как бесплатный **Personal** и платные **Pro**, **Team** и **Business**. Docker Personal подходит для индивидуальных разработчиков, образовательных учреждений и сообществ, которые работают с открытым исходным кодом, но связан с определенными ограничениями, с которыми можно ознакомиться здесь: docker.com/products/personal. Чтобы пользоваться этим планом, нужна учетная запись пользователя Docker, и в него включены настольный продукт Docker Desktop для Linux (и всех других основных платформ), а также движок Docker Engine для серверов.

Серверный движок Docker доступен для архитектур **x86** и **Arm** в виде пакетов в форматах `.deb` и `.rpm`, которые можно загрузить для Ubuntu, Debian, Fedora и CentOS. Для наших примеров мы будем использовать Docker Personal. В следующем разделе мы покажем, как установить Docker.

Как установить Docker

В зависимости от того, какой дистрибутив Linux вы используете, пакет Docker в официальном репозитории может быть устаревшим. В любом случае у вас есть два варианта: либо использовать пакет из репозитория вашего дистрибутива, либо загрузить последнюю доступную версию с официального сайта Docker.

Поскольку мы работаем со свежей системой, где раньше не был установлен Docker, нам не нужно беспокоиться о прежних версиях программного обеспечения и возможной несовместимости с новыми версиями. Мы будем использовать официальный репозиторий `apt` проекта Docker, который предоставляет самые свежие версии пакетов. Помните, что мы используем базовую систему Debian.

Чтобы установить Docker в Debian, можно поступить так:

1. Сначала обновите кеш репозитория и установите необходимые сертификаты для репозитория Docker, а также утилиту `curl`, которая понадобится на следующем шаге:

```
sudo apt update -y
sudo apt install ca-certificates curl
```

2. Чтобы использовать официальный репозиторий Docker, нужно добавить соответствующий ключ GPG. Для этого запустите такие команды:

```
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/debian/gpg -o /etc/apt/
keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc
```

3. Добавьте репозиторий:

```
echo \
"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.
asc] https://download.docker.com/linux/debian \
$(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

4. Теперь обновите список репозитория. В списке вы должны увидеть официальный репозиторий Docker:

```
packt@debian:~$ sudo apt update -y
# ... прочие репозитории ...
Get:4 https://download.docker.com/linux/debian bookworm InRelease [43.3 kB]
Get:5 https://download.docker.com/linux/debian bookworm/stable amd64 Packages
[38.1 kB]
```

5. Установите нужные пакеты Docker:

```
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-
plugin docker-compose-plugin
```

6. Чтобы убедиться, что вы установили пакеты из официального репозитория Docker, а не из репозитория Debian, запустите такую команду:

```
packt@debian:~$ sudo apt policy docker-ce
docker-ce:
  Installed: 5:28.0.4-1~debian.12~bookworm
  Candidate: 5:28.0.4-1~debian.12~bookworm
  Version table:
*** 5:28.0.4-1~debian.12~bookworm 500
    500 https://download.docker.com/linux/debian bookworm/stable amd64
Packages
  100 /var/lib/dpkg/status
  5:28.0.3-1~debian.12~bookworm 500
  500 https://download.docker.com/linux/debian bookworm/stable amd64
Packages
# ... и т. д. ...
```

Если в выводе фигурируют ссылки на загрузки с `docker.com`, это значит, что мы загрузили пакеты из официального репозитория Docker.

7. Проверьте состояние демона Docker. Если все в порядке, то он должен быть запущен сразу после установки:

```
sudo systemctl status docker
```

8. Во время установки создается группа `docker`. Чтобы в этом убедиться, можно просмотреть последние строки файла `/etc/group`:

```
tail /etc/group
```

9. Теперь нам понадобится учетная запись Linux, с помощью которой мы будем работать с Docker. Можно специально для этого создать нового пользовате-

ля, но в этом упражнении мы добавим в группу Docker свою собственную учетную запись:

```
sudo usermod -aG docker ${USER}
```

10. Проверьте, что пользователь успешно добавлен в группу. Для этого выйдите из системы, снова войдите и запустите такую команду:

```
groups
```

11. Вы завершили установку Docker. Теперь можно настроить демон Docker так, чтобы он запускался при старте системы:

```
sudo systemctl enable docker
```

Установка Docker — это только первый шаг. В следующем разделе вы узнаете о командах Docker, с помощью которых можно решать практические задачи.

Некоторые команды Docker

В интерфейсе командной строки Docker доступно множество подкоманд, и чтобы увидеть их список, запустите `docker --help`. В этом разделе мы не будем обсуждать все команды, а сосредоточимся только на тех, которые понадобятся, чтобы начать работать с Docker.

В первую очередь давайте проведем тест, чтобы убедиться, что система работает. Мы запустим команду `docker run`, чтобы проверить, можем ли мы получить доступ к Docker Hub и запустить контейнер оттуда. Проверочная команда выглядит так:

```
packt@debian:~$ sudo docker run hello-world
```

```
Unable to find image 'hello-world:latest' locally
# Не удалось найти локальный образ hello-world:latest
latest: Pulling from library/hello-world
# Получаем образ из library/hello-world
e6590344b1a5: Pull complete
Digest: sha256:7e1a4e2d11e2ac7a8c3f768d4166c2defeb09d2a750b010412b6ea13de1efb19
Status: Downloaded newer image for hello-world:latest
# Загружен новейший образ hello-world:latest
```

```
Hello from Docker!
```

```
# Вас приветствует Docker!
This message shows that your installation appears to be working correctly.
# Это сообщение свидетельствует о том,
# что ваша установка, скорее всего, работает правильно
# ... и т. д. ...
```

Это сообщение не требует дополнительных пояснений и служит жестом любезности со стороны команды Docker. После приветствия Docker простым и понятным языком объясняет, что команда делала в фоновом режиме, чтобы предо-

ставить вам это сообщение. Таким образом, запустив первую команду `docker run`, вы заодно узнали о рабочем процессе и убедились, что Docker успешно установлен. Это одна из основных команд Docker, которую вам предстоит использовать довольно часто.

Теперь давайте посмотрим, какие еще образы доступны на Docker Hub, — например, поищем образ Ubuntu, который можно запустить в виде контейнера. Для этого используйте команду `docker search`:

```
packt@debian:~$ docker search ubuntu
NAME                DESCRIPTION                                STARS
OFFICIAL
ubuntu              Ubuntu is a Debian-based Linux operating sys... 17516   [OK]
ubuntu/squid        Squid is a caching proxy for the Web. Long-t... 108
ubuntu/nginx        Nginx, a high-performance reverse proxy & we... 128
ubuntu/cortex       Cortex provides storage for Prometheus. Long... 4
# ... и т. д. ...
```

Эта команда перечисляет все образы, в названии которых встречается слово `ubuntu`. Вывод состоит из четырех столбцов:

NAME	Название образа
DESCRIPTION	Текстовое описание образа
STARS	Количество звезд (популярность образа по мнению пользователей)
OFFICIAL	Является ли этот образ официальным: поддерживает ли его компания, которая создала исходное ПО

Найдя нужный образ, его можно загрузить в свою систему с помощью команды `docker pull`. Давайте загрузим `ubuntu` — первый образ из списка на предыдущем снимке экрана:

```
docker pull ubuntu
```

В результате образ `ubuntu` будет загружен локально на ваш компьютер, и теперь из него можно будет запускать контейнеры. Чтобы узнать, какие образы уже доступны на вашем компьютере, запустите команду `docker images`:

```
packt@debian:~$ docker images
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
ubuntu        latest   a04dc4851cbc  8 weeks ago   78.1MB
hello-world   latest   74cc54e27dc4  2 months ago  10.1kB
```

Вас наверняка интересует, почему образ Ubuntu такой маленький. Дело в том, что образы Docker содержат только самые базовые пакеты, которые минимально необходимы для работы. В результате контейнер, который запускается из образа, чрезвычайно эффективно использует ресурсы.

В этом разделе мы продемонстрировали несколько самых элементарных команд, которые необходимы, чтобы начать использовать Docker. Теперь, когда вы умеете искать и загружать образы, давайте посмотрим, как управлять контейнерами Docker.

Как управлять контейнерами Docker

В этом разделе мы узнаем, как запускать, останавливать и удалять контейнеры Docker, а также выводить их список.

Как запускать контейнеры

Мы продолжим использовать образ Ubuntu, который только что загрузили. Чтобы его запустить, мы воспользуемся командой `docker run` с двумя ключами: `-i` для интерактивного вывода и `-t`, чтобы инициализировать псевдотерминал, который обеспечит интерактивный доступ к оболочке контейнера:

```
packt@debian:~$ docker run -it ubuntu
root@94ce5327179b:/#
```

Обратите внимание, что приглашение командной строки изменилось: теперь в нем отображается идентификатор контейнера, а пользователем по умолчанию является `root`. По сути вы теперь находитесь внутри образа Ubuntu, поэтому можете работать в командной строке Ubuntu обычным образом: например, обновлять репозиторий, устанавливать нужные приложения, удалять ненужные и т. д. Все изменения, которые вы вносите в образ контейнера, остаются в нем. Чтобы выйти из контейнера, просто введите `exit`.

Теперь мы покажем вам, как просматривать список контейнеров. Для этого упражнения, пожалуйста, не закрывайте терминал, в котором сейчас запущен контейнер с Ubuntu.

Как просматривать список контейнеров

Если вы откроете новый терминал в своей системе и запустите команду `docker ps`, она выведет список активных контейнеров Docker, которые запущены в данный момент. Вы увидите идентификатор контейнера `ubuntu`, который запущен в другом терминале, а также сведения о том, как давно создан контейнер и какие команды выполняются внутри него.

Команду `docker ps` можно запускать с различными ключами, например:

-a	--all	Отобразить все активные и неактивные контейнеры
-l	--latest	Отобразить последний созданный контейнер

Вот как выглядит запуск `docker ps` с разными ключами и без них:

```
packt@debian:~$ docker ps
```

```

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
94ce5327179b ubuntu "/bin/bash" 12 minutes ago Up 12 minutes flamboyant_pare

packt@debian:~$ docker ps -a
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
94ce5327179b ubuntu "/bin/bash" 12 minutes ago Up 12 minutes flamboyant_pare
df2bdf0736fe hello-world "/hello" 49 minutes ago Exited(0) 49 minutes ago jolly_ellis

packt@debian:~$ docker ps -l
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
94ce5327179b ubuntu "/bin/bash" 12 minutes ago Up 12 minutes flamboyant_pare

```

В выводе вы также видите имена, которые демон автоматически присвоил контейнерам случайным образом, например `flamboyant_pare` или `jolly_ellis`. Теперь давайте узнаем, как запускать, останавливать и удалять запущенные контейнеры.

Как запускать, останавливать и удалять контейнеры

Когда вы управляете контейнерами — например, запускаете или останавливаете их, на них можно ссылаться по идентификатору контейнера или по случайному имени Docker.

Чтобы запустить контейнер, используйте команду `docker start` с его именем или идентификатором, например:

```
packt@debian:~$ docker start flamboyant_pare
flamboyant_pare
```

Чтобы остановить контейнер, похожим образом используйте команду `docker stop`, например:

```
packt@debian:~$ docker stop flamboyant_pare
flamboyant_pare
```

Если вы остановите контейнер, а затем запустите команду `docker ps`, в списке больше не будет активных контейнеров.

Чтобы удалить контейнер, можно воспользоваться командой `docker rm`. Например, если вы хотите удалить исходный контейнер `hello-world` (в нашем случае он также называется `jolly_ellis`), используйте такую команду:

```
docker rm jolly_ellis
```

Как только вы удалите контейнер, будут потеряны все изменения, которые вы внесли в него и не зафиксировали.

Обратите внимание, что удаление контейнера не приведет к удалению исходного образа, который вы загрузили с Docker Hub.

Давайте посмотрим, как зафиксировать изменения, внесенные в контейнер. При этом вы фактически сохраняете определенное состояние контейнера в виде нового образа Docker.

Допустим, вы собираетесь разработать, протестировать и развернуть приложение на Python в контейнере Ubuntu. В стандартном образе Docker для Ubuntu не установлен Python. Давайте изменим и зафиксируем новый образ.

Как зафиксировать новый образ Docker

Упражнение этого раздела заключается в том, чтобы изменить существующий образ Ubuntu, установив пакеты Python, которые нужны для вашего приложения. Для этого выполните следующие шаги:

1. Сначала запустите контейнер и проверьте, установлен ли Python:

```
packt@debian:~$ docker run -it ubuntu

root@c50f0ea4f761:/# python --version
bash: python: command not found

root@c50f0ea4f761:/# python3 --version
bash: python3: command not found
```

2. Мы проверили Python 2 и Python 3, но оказалось, что ни та ни другая версия не установлена. Чтобы использовать последнюю версию Python, давайте установим ее с помощью стандартной команды `apt install`:

```
root@c50f0ea4f761:/# apt install python3
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
E: Unable to locate package python3    # Не удастся найти пакет python3
```

3. Команде `apt` не удалось найти пакет Python, потому что в минималистичном контейнере Docker нет списка репозиториев. Давайте обновим его:

```
root@6d0d22e6284c:/# apt update -y
Get:1 http://security.ubuntu.com/ubuntu noble-security InRelease [126 kB]
Get:2 http://archive.ubuntu.com/ubuntu noble InRelease [256 kB]
Get:3 http://archive.ubuntu.com/ubuntu noble-updates InRelease [126 kB]
# ... и т. д. ...
```

4. Как видите, теперь в контейнере доступен список репозиториев, и можно установить Python с помощью той же команды:

```
apt install python3
```

На этот раз команда не выдаст ошибок и выполнит установку.

5. Теперь, когда Python 3 установлен и в контейнер внесены необходимые изменения, экземпляр контейнера можно сохранить в новый образ Docker. Для этого запустите такую команду (в основной системе, а не в контейнере):

```
packt@debian:~$ docker commit -m "add python3 to ubuntu" -a "Alexandru
<info@packt.com>" 6d0d22e6284c packt/ubuntu-python3
sha256:2ca2e78017e17275acdf43b8550926f3d15805c8859b23c21e0e274155aa9661
```

Убедитесь, что в предыдущей команде указан правильный идентификатор контейнера (в нашем случае это `6d0d22e6284c`, но у вас будет другой идентификатор).

Мы использовали ключ `-m`, чтобы добавить комментарий, в котором описано, какие изменения мы внесли, а также ключ `-a`, чтобы указать автора коммита. Теперь команда `docker images` покажет новый образ Docker:

```
packt@debian:~$ docker images
REPOSITORY          TAG         IMAGE ID        CREATED         SIZE
packt/ubuntu-python3 latest      2ca2e78017e1   8 minutes ago  172MB
ubuntu              latest      a04dc4851cbc   2 months ago   78.1MB
hello-world         latest      74cc54e27dc4   2 months ago   10.1kB
```

Обратите внимание, что установка Python 3 более чем вдвое увеличила размер образа Docker.

В этом упражнении мы сохранили новый образ на своем локальном компьютере, но можно также загрузить его на Docker Hub, чтобы другие могли его использовать. Для этого вам понадобится учетная запись пользователя Docker.

К этому моменту вы уже научились использовать базовые команды Docker для того, чтобы открывать, запускать и сохранять контейнеры. В следующем разделе вы познакомитесь с Docker-файлами и научитесь создавать контейнерные образы.

Как работать с Docker-файлами

Прежде всего давайте разберемся, что такое Docker-файл. Это текстовый файл, который чаще всего состоит из инструкций о том, как автоматически собрать образ Docker. Инструкции строятся по общей схеме:

ИНСТРУКЦИЯ аргументы

Вот некоторые распространенные инструкции Docker-файлов:

FROM	Указывает образ, на основе которого создается сборка. Эта инструкция должна идти в начале Docker-файла
LABEL	Добавляет метаданные в формате <i>ключ=значение</i> , например описание или сведения о разработчике и версии образа. Метаданными не стоит злоупотреблять, потому что они увеличивают размер образа
RUN	Задаёт команды или сценарии, которые нужно запустить внутри образа

ADD	Копирует файлы или каталоги в файловую систему образа. Лучше всего подходит для того, чтобы добавлять в образ ресурсы из внешних источников
COPY	Подобно ADD, копирует файлы или каталоги. Лучше всего подходит для того, чтобы переносить файлы внутри файловой системы образа (например, при многоступенчатой сборке)
CMD	Задает команду, которая должна выполняться, когда из образа Docker запускается контейнер. В Docker-файле может быть не более одной инструкции CMD
USER	Задает учетную запись (а также, возможно, группу), от имени которой будут выполняться инструкции (такие, как RUN или CMD)
WORKDIR	Задает рабочий каталог по умолчанию для последующих команд внутри Docker-файла (таких, как RUN, CMD, COPY, ADD или ENTRYPOINT)
ENTRYPOINT	Настраивает контейнер так, чтобы он запускался как исполняемый файл

Полный список инструкций, которые доступны в Docker-файле, можно найти по адресу docs.docker.com/reference/dockerfile. В следующем разделе мы посмотрим, как с помощью Docker-файла собрать контейнерный образ.

Как создать контейнерный образ с помощью Docker-файла

В этом разделе мы разработаем Docker-файл, с помощью которого будет создаваться новый контейнерный образ Docker. Как в упражнении из раздела «Как зафиксировать новый образ Docker», мы подготовим образ для среды программирования Python. Чтобы создать Docker-файл, выполните такие операции:

1. Создайте новый каталог внутри домашнего каталога и перейдите в него:

```
mkdir ~/my_docker_images && cd ~/my_docker_images
```
2. Создайте в текущем каталоге новый Docker-файл:

```
touch py_env_dockerfile
```
3. Откройте этот Docker-файл в текстовом редакторе и внесите в него такое содержание:

```
FROM debian

RUN apt update -y
RUN apt upgrade -y
RUN apt install python3 -y
```
4. Обратите внимание, что на этот раз мы использовали в качестве базового образа не ubuntu, а debian, который можно найти с помощью команды `docker search debian`. В инструкциях Docker-файла мы воспроизвели ту же про-

цедуру, которую до этого выполняли вручную в разделе «Как зафиксировать новый образ Docker».

5. Теперь, когда Docker-файл готов, давайте запустим команду `docker build`, чтобы создать новый образ Docker:

```
packt@debian:~/my_docker_images$ docker build ~/my_docker_images -f py_env_
dockerfile -t pydeb
[+] Building 2.5s (8/8) FINISHED docker:default
=> [internal] load build definition from py_env_dockerfile      0.0s
=> => transferring dockerfile: 121B                             0.0s
=> [internal] load metadata for docker.io/library/debian:latest 2.5s
=> [internal] load .dockerignore                               0.0s
=> => transferring context: 2B                                    0.0s
=> [1/4] FROM docker.io/library/debian:latest@sha256:18023f131f5... 0.0s
=> CACHED [2/4] RUN apt update -y                               0.0s
=> CACHED [3/4] RUN apt upgrade -y                              0.0s
=> CACHED [4/4] RUN apt install python3 -y                      0.0s
=> exporting to image                                           0.0s
=> => exporting layers                                           0.0s
=> => writing image sha256:52d0025a7489813e0f98be358aaedff0bcf59... 0.0s
=> => naming to docker.io/library/pydeb                          0.0s
```

Здесь с помощью ключа `-f` мы указали имя Docker-файла, а с помощью ключа `-t` — имя образа, который мы хотим создать (в нашем случае `pydeb`). Вывод показывает, что сборка прошла успешно.

6. С помощью `docker images` можно убедиться, что новый образ `pydeb` благополучно создан:

```
packt@debian:~/my_docker_images$ docker images
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
pydeb	latest	52d0025a7489	20 minutes ago	189MB
packt/ubuntu-python3	latest	2ca2e78017e1	2 hours ago	172MB
ubuntu	latest	a04dc4851cbc	2 months ago	78.1MB
hello-world	latest	74cc54e27dc4	2 months ago	10.1kB

7. Теперь из нового образа `pydeb` можно запустить контейнер с помощью такой команды:

```
docker run -it pydeb
```

8. Чтобы убедиться, что контейнер работает, откройте новое окно терминала и запустите команду `docker ps`:

```
packt@debian:~$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
e02408135428	pydeb	"bash"	About a minute ago	Up	About a minute	relaxed_perlman

К этому моменту вы уже достаточно знаете о Docker, чтобы уверенно использовать контейнеры в среде эксплуатации. В следующем разделе мы покажем, как с помощью Docker развернуть очень простое приложение — элементарный статический сайт.

Как разворачивать контейнерные приложения с помощью Docker

До сих пор мы говорили о том, как использовать Docker и как управлять контейнерами. Docker предоставляет гораздо более широкие возможности, и мы надеемся, что примеров, которые мы продемонстрировали до сих пор, будет достаточно, чтобы вы заинтересовались этой темой и захотели узнать больше. Docker — отличный инструмент для разработчиков, потому что с его помощью можно разворачивать приложения упрощенным способом, при котором не требуется воссоздавать среду разработки. В следующем разделе мы рассмотрим, как на основе Docker развернуть простой сайт.

Как развернуть сайт с помощью Docker

Чтобы развернуть сайт с помощью Docker, выполните следующие действия:

1. Мы будем использовать бесплатный шаблон сайта, который можно загрузить из интернета (free-css.com/free-css-templates/page262/focus). Давайте загрузим шаблон в наш домашний каталог с помощью утилиты `wget`:

```
wget https://www.free-css.com/assets/files/free-css-templates/download/page262/focus.zip
```

2. Шаблон представляет собой сжатый файл ZIP, поэтому нам придется его распаковать. Для этого мы воспользуемся командой `unzip`:

```
unzip focus.zip -d ~/my_docker_images/docker_webapp/
```

3. С помощью ключа `-d` мы распаковали файл в каталог `docker_webapp`, который был создан внутри существующего каталога `my_docker_images`. В итоге готовые файлы нашего сайта находятся здесь:

```
~/my_docker_images/docker_webapp/focus
```

4. После того как шаблон распакован, давайте создадим в текущем рабочем каталоге Docker-файл `webapp_dockerfile`, в котором будет всего две строки:

```
FROM nginx
COPY . /usr/share/nginx/html
```

Вот что делают эти инструкции:

- В первой строке инструкция `FROM` задает базовый образ, на основе которого мы построим собственный образ. В данном случае мы используем официальный образ популярного веб-сервера `nginx`, который доступен на Docker Hub. (Подробнее об `nginx` см. в главе 13.)
 - Во второй строке инструкция `COPY` задает место в файловой системе будущего образа, куда будет скопировано содержимое сайта.
5. После этого можно создать образ Docker с помощью команды `docker build`:

```
docker build ~/my_docker_images/docker_webapp/focus -f webapp_dockerfile -t webapp
```

6. Запустите команду `docker images`, чтобы убедиться, что создан новый образ, который называется `webapp`:

```
packt@debian:~$ docker images
REPOSITORY          TAG          IMAGE ID          CREATED           SIZE
webapp              latest       70d163df9c81     About a minute ago 194MB
# ... и т. д. ...
```

7. Теперь можно запустить новый контейнер из нового образа `webapp`. Поскольку нам нужно будет обращаться к контейнеру извне, понадобится открыть определенные порты. Ключ `-p 8080:80` команды `docker run` отобразит порт `80` контейнера на порт `8080` базовой системы, а с помощью ключа `-d` мы соединим контейнер и запустим его в фоновом режиме. Итоговая команда выглядит так:

```
packt@debian:~$ docker run -it -d -p 8080:80 webapp
4a9ca78fdb152fe9dc41d1dbbd32f140704037673ed17ada693dfba82f1f5f8
```

Мы могли бы использовать порт `80` как на базовом узле, так и в контейнере, но на базовом узле он может быть занят другими службами.

8. Теперь, чтобы вызвать новое контейнерное приложение, можно зайти в браузер и набрать в адресной строке локальный IP-адрес базового узла и порт: в нашем случае это `http://192.168.122.48:8080`¹. Вот как выглядит наш сайт, который развернут в Docker:

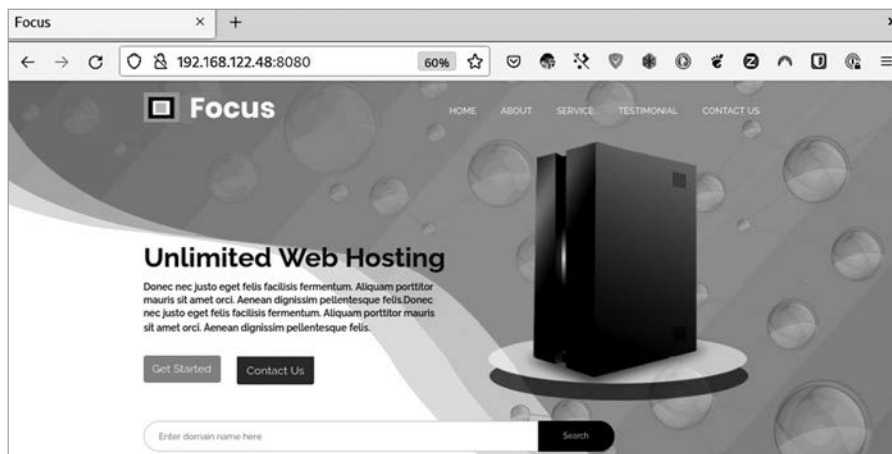


Рис. 12.5. Запуск контейнерного веб-приложения в браузере

Можно убедиться, что сайт доступен с нашего локального узла. О том, как развернуть сайт на виртуальном частном сервере, см. главу 13.

¹ В Debian-подобных системах нет работающего сетевого экрана. Если будете использовать Centos-подобные системы, то вам следует предварительно разрешить tcp-доступ к порту 8080. — *Примеч. ред.*

Итоги

В этой главе мы рассказали о том, как важна контейнеризация. Вы узнали, что такое контейнеры, как они работают и чем они полезны. Контейнеры — это основа нынешней революции в области DevOps, и теперь вы готовы их использовать. Мы также рассказали о Docker и его основных командах, так что вы получили необходимые основы для того, чтобы познакомиться с облачными службами. Виртуализация и контейнерные технологии лежат в основе облачных и серверных решений.

В следующей главе мы покажем, как устанавливать и настраивать различные серверы на базе Linux — например, серверы DNS, DHCP и почтовые серверы.

Вопросы

Вот вопросы, с помощью которых можно закрепить основные понятия, о которых шла речь в этой главе:

1. В чем основная разница между контейнерами и виртуальными машинами?
Подсказка: обратитесь к рис. 12.1.
2. Как работают контейнерные технологии?
3. Каковы два основных компонента архитектуры Docker?
4. Какая команда Docker показывает запущенные контейнеры?
5. Какая команда позволяет управлять сетевыми службами в контейнерах?

Подсказка: обратитесь к справке по команде `docker network`.

Дополнительная литература

Чтобы получить дополнительную информацию по темам, которые рассматривались в этой главе, можно обратиться к следующим изданиям Packt:

- Эрл Вод (Earl Waud). «Docker Quick Start Guide».
- Расс Маккендрики (Russ McKendrick). «Mastering Docker», Fourth Edition.
- Константин Иванов. «Containerization with LXC».
- Эммануил Гадзурас (Emmanouil Gkatzouras). «A Developer's Essential Guide to Docker Compose»¹.

¹ Гадзурас Э. «Docker Compose для разработчика. Упростите разработку и оркестрацию многоконтейнерных приложений».

13

Как настраивать серверы Linux

В этой главе вы узнаете, как настраивать различные типы серверов Linux — **DNS**, **DHCP**, **NFS**, а также файловые серверы **Samba**. Все эти серверы так или иначе лежат в основе Всемирной паутины. Хотя мы не будем подробно останавливаться на этом вопросе, вам следует знать, что ваш компьютер показывает точное время благодаря грамотно реализованному серверу **NTP**. А благодаря эффективным серверам **DHCP**, веб-серверам и файловым серверам вы можете совершать покупки в интернете и обмениваться файлами с друзьями и коллегами. Любому системному администратору Linux надо уметь конфигурировать различные службы, которые обеспечивают работу всех этих серверов. В книге мы рассмотрим только избранные серверы Linux, те, которые считаем наиболее важными на сегодняшний день. Чтобы получить дополнительную информацию, пожалуйста, обратитесь к разделу «Дополнительная литература» в конце главы.

В этой главе мы рассмотрим такие темы:

- Введение в службы Linux.
- Как настраивать серверы:
 - SSH;
 - DNS;
 - DHCP;
 - NFS;
 - файловый сервер Samba.

Технические требования

Чтобы осваивать материал этой главы, нужно в общих чертах представлять себе, как работают сетевые технологии, и знать основные команды Linux. Вам потребуется доступ к нескольким рабочим системам, которые размещены локально или в облаке. Если такого доступа нет, вы можете использовать локальные

виртуальные машины на своем компьютере. Кроме того, хорошо, если у вас будет доступ к собственному домену, который пригодится для упражнений. Для примеров в этой главе мы будем использовать Ubuntu Server 22.04.2 LTS, но также отлично подойдет любой другой основной дистрибутив Linux — например, Fedora, RHEL, openSUSE или Debian.

Введение в службы Linux

Все, что вы узнали до сих пор, можно с легкостью применить к любой рабочей станции, настольному компьютеру или ноутбуку под управлением Linux. Мы углубились в продвинутые вопросы сетевых технологий, которые облегчат вам путь к тому, чтобы стать опытным системным администратором. Теперь мы перейдем на территорию *серверов*, через которую лежит дорога к *облачным технологиям*, — о них подробно поговорим в последних четырех главах этой книги.

Сервер Linux, в отличие от рабочей станции, — это система, которая обеспечивает доступ к тому или иному контенту по сети. При этом сервер предоставляет свои аппаратные и программные ресурсы в распоряжение клиентов, которые к нему обращаются. Например, всякий раз, когда вы вводите адрес сайта в адресную строку браузера, браузер обращается к серверу, а именно — к **веб-серверу**. Когда вы распечатываете документ на офисном принтере, вы обращаетесь к **серверу печати**, а когда читаете электронную почту — к **почтовому серверу**. Все это специализированные системы, которые работают под управлением особого программного обеспечения (так называемых служб) и предоставляют клиенту (то есть вам) запрошенные материалы. Обычно серверы — это мощные системы с внушительным объемом ресурсов, которые обслуживают запросы клиентов.

С другой стороны, **рабочие станции** нередко тоже оснащены мощным оборудованием, но обычно используются для персональной работы, а не для доступа клиентов по сети. Содержание этой и последующих глав в первую очередь относится к серверным системам, хотя не ограничивается ими.

Вероятно, вы уже в той или иной форме сталкивались с настройкой различных серверов Linux — таких как веб-сервер, файловый сервер или сервер электронной почты — и, возможно, задавались вопросом, почему они так называются. Это не аппаратные комплексы, которые обычно ассоциируются с «настоящими» серверами, а просто службы, которые работают под управлением Linux. В свою очередь, службы — это программы, которые функционируют в фоновом режиме и известны в мире Linux как **демоны**. Вы кратко познакомились с демонами и подсистемой инициализации в главе 5 «Как работать с процессами, демонами и сигналами». Предком всех процессов является процесс инициализации, который стартует одним из первых при загрузке Linux. Современные дистри-

бутивы (в том числе Ubuntu, CentOS, Fedora, openSUSE и др.) используют `systemd` в качестве подсистемы инициализации по умолчанию.

В этой главе вам предстоит запускать некоторые элементарные команды для работы со службами в Linux. Если вам нужно освежить свои знания, обратитесь к главе 5.

Первая команда, которую мы вспомним, — это `ps`. С ее помощью можно, например, продемонстрировать запущенный процесс инициализации:

```
packt@ubuntu:~$ ps -fp 1
UID PID PPID C STIME TTY TIME CMD
root 1 0 0 17:52 ? 00:00:01 /usr/lib/systemd/systemd --systemd
--deserialize=56
```

Здесь ключ `-p` выводит сведения о процессе с определенным PID (в данном случае 1), а ключ `-f` отображает данные в расширенном формате.

Как правило, первым в списке процессов идет `systemd`. Иногда в старых операционных системах из соображений обратной совместимости используется имя `init`, но на самом деле это псевдоним `systemd`. Если ввести в командной строке `man init`, вы увидите справочную страницу для `systemd`. Будучи родителем всех служб, `systemd` запускает все процессы параллельно, чтобы оптимизировать загрузку и заставить службы быстрее отвечать на запросы. Чтобы узнать, сколько времени заняла загрузка, можно запустить команду `systemd-analyze`:

```
packt@ubuntu:~$ systemd-analyze
Startup finished in 2.656s (kernel) + 3.457s (userspace) = 6.114s
# Начальная загрузка заняла 2,656 с в пространстве ядра
# + 3,457 с в пользовательском пространстве = 6,114 с
graphical.target reached after 3,434s in userspace.
# Система готова к взаимодействию с пользователем через 3,434 с
# в пользовательском пространстве
```

Как вы знаете, `systemd` управляет **модулями**; эти системные ресурсы бывают различных типов — например, служба (`.service`), точка монтирования (`.mount`) или сокет (`.socket`). Модули запускает уже знакомая вам команда `systemctl` — основная утилита командной строки для работы со службами (демонами) `systemd` в Linux.

Чтобы узнать, сколько времени было затрачено на запуск каждого модуля, можно использовать команду `systemd-analyze blame`:

```
packt@ubuntu:~$ systemd-analyze blame
1.708s systemd-networkd-wait-online.service
558ms dev-mapper-ubuntu\x2d\x2dvg\x2dubuntu\x2d\x2dlv.device
515ms snapd.seeded.service
466ms snapd.service
# ... и т. д. ...
85ms systemd-udevd.service
8ms boot-efi.mount
8ms snapd.socket
```

Здесь показано только начало и конец вывода, и можно видеть различные типы устройств: `.service`, `.device` и т. д. Чтобы подробнее узнать о команде `systemctl`, обратитесь к справочным страницам или перечитайте главу 5.

Надеемся, что это краткое введение в службы Linux позволило вам повторить соответствующий материал главы 5 в достаточной степени для того, чтобы начать настраивать и конфигурировать конкретные службы Linux. В следующих разделах мы поговорим о том, как управлять некоторыми из наиболее важных служб, такими как **SSH**, **DNS**, **DHCP**, **Samba** и **NFS**. Пришло время засучить рукава и наладить их самостоятельно. Сначала мы посмотрим, как настроить сервер SSH в Ubuntu.

Как настроить сервер SSH

Мы настроим SSH на компьютере под управлением Ubuntu Server 22.04.2 LTS. На протяжении всей книги мы уже несколько раз использовали соединения SSH; в частности, в главе 11 «Как работать с виртуальными машинами» вы узнали, как создать пару ключей SSH, которые в тот момент были нужны для `cloud-init`. На этот раз мы поговорим о том, как установить **OpenSSH**, как включить SSH и как изменить некоторые его настройки.

Как установить и настроить OpenSSH в Ubuntu

Чтобы использовать SSH, первым делом нужно установить соответствующий пакет:

```
sudo apt install openssh-server
```

Скорее всего, `openssh-server` уже установлен в вашей системе. Если это так, можно пропустить этот шаг и перейти к конфигурационным файлам.

После установки можно включить и запустить службу `ssh` с помощью таких команд:

```
sudo systemctl enable ssh && sudo systemctl start ssh
```

Конфигурационный файл `openSSH` находится в расположении `/etc/ssh/sshd_config`. По умолчанию в нем уже много настроек, так что можно открыть файл в текстовом редакторе и изменить те параметры, которые нас интересуют. Не забудьте раскомментировать строки, которые мы будем редактировать (то есть удалить `#` в начале строк). В типичную конфигурацию SSH входят такие настройки:

- Запретить удаленно подключаться по SSH от имени пользователя `root`:

```
PermitRootLogin no
```

По умолчанию для этого параметра задано значение `prohibit-password`, которое запрещает пользователю `root` входить в систему с помощью клави-

атуры (например, вводя пароль). Мы изменили значение на `no`, чтобы пользователь `root` вообще не смог подключиться по SSH.

- Отключить аутентификацию по паролю SSH; для этого нужно задать значение `no` в следующих строках:

```
PasswordAuthentication no
PermitEmptyPasswords no
```

Сохраняйте эти изменения только после того, как скопируете свою пару ключей на удаленную машину и убедитесь, что можете подключиться с ее помощью. В противном случае вы потеряете доступ к своему серверу.

- Чтобы разрешить аутентификацию с открытым ключом, раскомментируйте эту строку:

```
PubkeyAuthentication yes
```

- На всякий случай напомним команды, которые использовались в главе 11, чтобы включить аутентификацию с открытым ключом:

```
ssh-keygen
ssh-copy-id имя_пользователя@узел
```

Не забывайте, что `ssh-keygen` можно использовать с различными ключами командной строки: например `-b` (разрядность) или `-t` (алгоритм шифрования). После того как вы настроите ключи, вы сможете подключиться к серверу с помощью команды `ssh`, например:

```
ssh packt@192.168.0.113
```

Здесь представлены только самые простые параметры конфигурации, которые необходимы, чтобы начать использовать SSH, работая с удаленной системой или виртуальной машиной. Однако OpenSSH — это очень мощный инструмент, который предлагает множество возможностей. Чтобы изучить их подробнее, обратите внимание на две ссылки:

- Документация по серверу OpenSSH на официальном сайте Ubuntu: documentation.ubuntu.com/server/how-to/security/openssh-server.
- Официальные справочные руководства проекта OpenSSH: openssh.com/manual.html.

В следующем разделе мы покажем, как настроить сервер DNS.

Как настроить сервер DNS

Одна из наиболее широко используемых служб DNS называется **BIND 9**; ее официальный сайт — isc.org/bind. Прежде чем продолжить, давайте определимся, что именно мы собираемся настраивать. На нашем компьютере под управлением Ubuntu Server 22.04.2 LTS мы создадим серверы двух типов: **кэширу-**

ющий сервер имен и первичный сервер имен, с помощью которых вы сможете управлять именами узлов и частными IP-адресами в своей локальной сети.

Серверы DNS делятся на разные функциональные типы: например, бывают **авторитетные, кэширующие** или **переадресующие** серверы. Кэширующий сервер DNS всегда отвечает на рекурсивные запросы клиентов. Кроме того, серверы DNS бывают первичными и вторичными: это разновидности авторитетных серверов, которые во многом похожи, но различаются тем, откуда они получают информацию о зонах. Чтобы подробнее узнать о DNS, можно обратиться к серии учебных пособий по адресу digitalocean.com/community/tutorial-series/an-introduction-to-managing-dns.

Существуют и другие сценарии настройки DNS, но для того, чтобы познакомиться с основами, примеров из этого раздела будет достаточно. Если вы захотите настроить вторичный сервер DNS, для этого понадобится еще одна система, а если вы используете **VPS**, они должны находиться в одном центре обработки данных и в одной локальной сети. Впрочем, в нашем случае мы ограничимся локальной системой в нашей небольшой частной сети.

Сначала мы установим пакет `bind9` в Ubuntu с помощью такой команды:

```
sudo apt install bind9
```

После того как пакеты установлены, можно проверить, работает ли BIND. Для этого запустите команду `nslookup`, чтобы получить записи DNS для домена `google.com` с локального сервера (`127.0.0.1`):

```
packt@nuc5:~$ nslookup google.com 127.0.0.1
Server:      127.0.0.1
Address:     127.0.0.1#53

Non-authoritative answer:
Name:   google.com
Address: 64.233.163.101
Name:   google.com
# ... и т. д. ...
```

После того как мы убедились, что служба работает, ее можно начинать настраивать. Сначала мы сконфигурируем кэширующую службу DNS, но перед этим советуем вам сделать резервные копии конфигурационных файлов `/etc/bind/named.conf`, `/etc/bind/named.conf.options`, `/etc/hosts` и `/etc/resolv.conf`.

Как настроить кэширующий сервер DNS

По умолчанию BIND 9 работает как кэширующий сервер, а это значит, что настроить его будет довольно просто. Мы немного подправим конфигурационный файл, чтобы служба работала в соответствии с нашими требованиями:

1. Для начала разрешите службу BIND 9 в межсетевом экране с помощью такой команды:

```
sudo ufw allow Bind9
```

2. В Ubuntu вам понадобится отредактировать конфигурационный файл `/etc/bind/named.conf.options`. В этом файле уже находятся некоторые настройки по умолчанию и несколько закомментированных строк (они начинаются с `//`) с подробным описанием того, как использовать этот файл. Все изменения нужно будет вносить внутри директивы `options` (между фигурными скобками), потому что это единственная директива, которая существует по умолчанию.

Чтобы упростить настройку, сначала отключим IPv6 и будем использовать только **IPv4**. Для этого закомментируйте следующую строку, добавив в ее начало две косые черты:

```
// listen-on-v6 { any; };
```

3. Внутри директивы `forwarders` вам понадобится отредактировать список IP-адресов. Эта директива указывает серверу, где искать адреса, которых нет в локальном кэше. Для простоты мы добавим публичные серверы DNS Google, но вы можете также использовать серверы вашего **интернет-провайдера**. В итоге директива `forwarders` должна выглядеть так:

```
forwarders {  
    8.8.8.8;  
    8.8.4.4;  
};
```

4. Можно также добавить директиву `allow-query`, которая указывает, с каких адресов сервер может принимать запросы DNS. Например, добавим адрес своей локальной сети:

```
allow-query {  
    localhost;  
    192.168.0.0/24;  
};
```

5. Сохраните файл и закройте редактор. Если хотите, проверьте конфигурацию BIND 9 с помощью команды `named-checkconf`. Если эта команда ничего не выводит, значит, в конфигурации нет синтаксических ошибок. Перезапустите службу BIND 9 и на всякий случай проверьте ее состояние.

6. Если BIND 9 работает нормально, то с любого другого компьютера в сети к ней можно направить запрос `nslookup`, например:

```
alexandru@debian:~$ nslookup google.com 192.168.126.128  
Server:      192.168.126.128  
Address:     192.168.126.128#53
```

```
Non-authoritative answer:  
Name:   google.com  
Address: 142.250.186.110
```

Мы запустили эту команду на другом компьютере в той же сети и указали IP-адрес того узла, на котором мы настраивали сервер DNS. Результат показывает, что служба DNS на нашей тестовой машине работает нормально.

Теперь у вас есть рабочий кэширующий сервер DNS в частной сети. В следующем разделе мы посмотрим, как создать первичный сервер DNS.

Как настроить первичный сервер DNS

Чтобы настроить первичный сервер DNS, нам понадобится доменное имя, которое он будет обслуживать. В этом разделе мы будем использовать имя `calcatinge.ro`, но когда вы будете выполнять упражнения в своей системе, пожалуйста, подставляйте вместо него собственный домен. Нам понадобится создавать новые зоны для конфигурации BIND 9, и мы будем добавлять информацию о них в файл `/etc/bind/named.conf.local`. Давайте создадим новую зону для нашего домена `calcatinge.ro`.

Что такое **зона DNS**? Короткий ответ заключается в том, что это область пространства доменных имен, которую обслуживает конкретная организация или администратор. Зоны также предлагают гранулированный подход к администрированию различных компонентов. Подробнее о зонах DNS можно узнать на странице ibm.com/think/topics/dns-zone.

Вот содержимое нашего конфигурационного файла:

```
//
// Do any local configuration here
//

// Consider adding the 1918 zones here, if they are not used in your
// organization
//include "/etc/bind/zones.rfc1918";

zone "calcatinge.ro" {
    type primary;
    file "/etc/bind/db.calcatinge.ro";
    allow-transfer { 192.168.0.113; };
    also-notify { 192.168.0.113; };
};
```

Вот что означают параметры директивы `zone`:

<code>"calcatinge.ro"</code>	Имя домена, который будет обслуживать зона
<code>type</code>	Тип зоны — в данном случае <code>primary</code> (первичная), но бывают и другие типы — например, <code>secondary</code> , <code>forward</code> или <code>hint</code>
<code>file</code>	Путь к фактическому файлу зоны, который мы создадим позже
<code>allow-transfer</code>	Список узлов, которым разрешено передавать данные зоны с этого сервера
<code>also-notify</code>	Список узлов, которые будут уведомляться об изменениях зоны

Теперь давайте создадим новую зону для нашего домена:

1. Скопируйте файл `/etc/bind/db.local` в тот же каталог под именем `db.calcatinge.ro`. Это будет наш файл зоны:

```
packt@nuc5:/etc/bind$ sudo cp db.local db.calcatinge.ro
```

2. Теперь откройте файл `db.calcatinge.ro` в текстовом редакторе и внесите в него IP-адрес, доменное имя и другие сведения о своей доменной зоне по образцу того, как мы сделали это для домена `calcatinge.ro` на своем компьютере:

```
;
; BIND data file for local loopback interface
;
$TTL      604800
@         IN      SOA      ns.calcatinge.ro. admin.calcatinge.ro. (
                                2           ; Serial
                                604800      ; Refresh
                                86400       ; Retry
                                2419200     ; Expire
                                604800 )   ; Negative Cache TTL
;
@         IN      NS       ns.calcatinge.ro.
@         IN      A        191.168.0.113
ns        IN      A        191.168.0.113
```

3. Записи DNS представлены в конце файла. Вот что они означают:
 - Таблица зоны DNS состоит из четырех столбцов: имя узла, класс записи DNS, тип записи и значение.
 - Имя узла `@` означает, что запись относится к текущему домену (в данном случае `calcatinge.ro`). Имя `ns` относится к записи для домена `ns.calcatinge.ro`.
 - Класс записи `IN` указывает на то, что сеть является интернетом¹.
 - Существует много типов записей DNS, например:

A	Адрес узла IPv4
NS	Адрес авторитетного сервера доменных имен
CNAME	Каноническое имя (псевдоним доменного имени)
TXT	Текстовые данные ²

¹ Другие классы DNS фактически вышли из употребления и не встречаются в современных записях DNS. — *Примеч. ред.*

² Изначально тип записи TXT предназначался для того, чтобы включить в зону DNS произвольные дополнительные данные, которые мог бы прочитать человек. В современном DNS такие записи в основном служат для машиночитаемых данных, которые не описываются другими типами записей. — *Примеч. ред.*

SOA	Основные сведения о зоне DNS, в том числе имя первичного сервера имен, серийный номер домена и различные таймеры (например, частота обновления зоны)

- Значение в последнем столбце чаще всего содержит IP-адрес или имя узла.
4. После того как вы сохраните обновленный файл `db.calcatinge.ro`, нужно будет перезапустить службу **RNDC** — утилиту в составе BIND, которая управляет сервером имен:

```
sudo rndc reload
```

5. Теперь можно проверить, работает ли первичный сервер DNS. Попробуйте выполнить команду `nslookup` с другой системы в сети (не забывайте всюду указывать свои имена и IP-адреса вместо наших):

```
nslookup calcatinge.ro 192.168.0.113
```

Вывод этой команды практически наверняка покажет, что на локальном сервере DNS, который находится по указанному IP-адресу, есть рабочий файл зоны. Ваш первичный сервер DNS благополучно работает — правда, только в локальной сети.

На случай, если с первичным сервером DNS что-то произойдет, всегда полезно иметь вторичный сервер, и в следующем разделе мы покажем, как это сделать.

Как настроить вторичный сервер DNS

Неудивительно, что вторичный сервер DNS должен быть установлен не на том же оборудовании, что первичный, хотя его можно разместить в той же сети. Если вы пользуетесь услугами центра обработки данных, задействуйте еще один VPS в той же сети, что и первый. Если вы практикуетесь в домашней частной сети, убедитесь, что в вашем распоряжении есть еще одна система.

Мы будем демонстрировать упражнения на примере еще одной системы NUC, которая тоже работает под управлением Ubuntu Server 22.04.2 LTS. Для конфигурации нам понадобится ее IP-адрес, и в нашем случае это `192.168.0.140`. На этой, второй машине тоже придется установить и настроить BIND 9. Прежде чем налаживать вторичный сервер DNS, нужно изменить конфигурацию первичного сервера.

Как изменить конфигурационные файлы первичного сервера DNS

Чтобы изменить конфигурацию первичного сервера DNS и разрешить ему передавать сведения о зоне на вторичный сервер, выполните такие действия:

1. Откройте конфигурационный файл `/etc/bind/named.conf.local` и добавьте IP-адрес второго сервера внутри директив `allow-transfer` и `also-notify`:

```
allow-transfer { 192.168.0.113; 192.168.0.140; };
also-notify { 192.168.0.113; 192.168.0.140; };
```

2. Сохраните файл и перезапустите службу BIND 9.
3. Теперь откройте конфигурационный файл `/etc/bind/named.conf.options` и добавьте в него список доступа доверенных IP-адресов. В нашем случае это те же адреса первичного и вторичного серверов DNS. Это делается с помощью директивы `acl`, которую нужно вставить *перед* уже существующей директивой `options`:

```
acl "trusted"1 {
    192.168.0.113;
    192.168.0.140;
};

options {
    ...
};
```

4. Также нужно добавить новые директивы в конец блока `options`, перед закрывающей фигурной скобкой:

```
options {

    // ... существующие директивы ...

    recursion yes;
    allow-recursion { trusted ; };
    listen-on { 192.168.0.140; };
    allow-transfer { none; };
};
```

Давайте разберемся, что означают эти строки. Директива `recursion yes` разрешает рекурсию и кэширование на сервере, а `allow-recursion` указывает, от каких клиентов можно принимать рекурсивные запросы; в данном случае эти клиенты представлены списком доступа `trusted`. Директива `listen-on` задает IP, на котором сервер прослушивает запросы DNS, а благодаря `allow-transfer` со значением `none` мы запретили другим узлам передавать сведения о зоне с текущего сервера. Подробнее о параметрах конфигурации BIND 9 можно узнать в справочнике bind9.readthedocs.io.

Рекурсия — это такой способ разрешения доменных имен, при котором компьютер ищет IP-адрес по доменному имени, обращаясь сначала к данным локального кэша и локальному серверу DNS. Если адрес не найден, сервер DNS обращается к другому известному серверу DNS, и так до тех пор, пока имя не будет разрешено.

¹ В данном случае `trusted` (доверенные) — это искусственно придуманное имя, по которому мы впоследствии будем обращаться к этому списку доступа. — *Примеч. ред.*

5. Настройка завершена, так что теперь следует перезапустить службу BIND 9:

```
sudo systemctl restart bind9
```

Далее давайте узнаем, как настроить вторичный сервер.

Как настроить вторичный сервер DNS

Как уже говорилось, на вторичном сервере тоже нужно установить BIND 9, а после этого следует выполнить такие шаги:

1. Перейдите в файл `/etc/bind/named.conf.options` и добавьте перед уже существующей директивой `options` такую же директиву `acl`, как на первичном сервере:

```
acl "trusted" {  
    192.168.0.113;  
    192.168.0.140;  
};
```

```
options {  
    ...  
}
```

2. Также добавьте уже знакомые строки внутри директивы `options`:

```
options {  
  
    // ... существующие директивы ...  
  
    recursion yes;  
    allow-recursion { trusted ; };  
    listen-on { 192.168.0.140; };  
    allow-transfer { none; };  
  
    forwarders {  
        8.8.8.8;  
        8.8.4.4;  
    };  
};
```

3. Теперь отредактируйте файл `/etc/bind/named.conf.local` и добавьте нужные зоны, но на этот раз укажите тип `secondary`, а не `primary`, как на первичном сервере.

Вот как должна выглядеть директива `zone` в конфигурационном файле вторичного сервера DNS; сравните ее с аналогичной директивой первичного сервера (в начале подраздела «Как настроить первичный сервер DNS»):

```
zone "calcatinge.ro" {  
    type secondary;  
    file "/etc/bind/db.calcatinge.ro";  
    primaries { 192.168.0.113; };  
};
```

- Осталось разрешить запросы DNS в межсетевом экране для вторичного сервера и перезапустить службу BIND 9:

```
sudo ufw allow Bind9
sudo systemctl restart bind9
```

Теперь у вас настроены и работают два сервера DNS: первичный и вторичный. В следующем разделе мы покажем, как настроить локальный сервер DHCP.

Как настроить сервер DHCP

DHCP — это сетевая служба, которая назначает IP-адреса узлам в сети; она делает это автоматически, без управления со стороны узлов. Чаще всего сервер DHCP предоставляет клиентам IP-адреса и маски подсетей, IP-адрес шлюза по умолчанию и IP-адрес сервера DNS.

Чтобы установить службу DHCP в Ubuntu, запустите такую команду:

```
sudo apt install isc-dhcp-server
```

Мы будем выполнять упражнения на той же системе, на которую установили службы DNS в предыдущем разделе. После установки мы отредактируем два конфигурационных файла: конфигурация по умолчанию находится в файле `/etc/dhcp/dhcpd.conf`, а интерфейсы настраиваются в файле `/etc/default/isc-dhcp-server`.

- Сначала давайте настроим простейший локальный сервер DHCP. Для этого в файл `/etc/dhcp/dhcpd.conf` понадобится добавить пул IP-адресов. Можете либо раскомментировать одну из директив `subnet`, которые уже есть в файле, либо создать новую, что мы и сделаем. Наша существующая подсеть — `192.168.0.0/24`, и мы добавим соответствующие адреса в конфигурацию сервера DHCP. (Чтобы освежить свои знания о сетевых технологиях, обратитесь к главе 7 «Сетевое окружение Linux».)

```
subnet 192.168.1.0 netmask 255.255.255.0 {
    range 192.168.1.100 192.168.1.200;
    option routers 192.168.1.1;
    option domain-name-servers 192.168.1.2, 192.168.1.3;
    option domain-name "homecomputer.local";
    option broadcast-address 192.168.1.255;
}
```

- В том же файле `/etc/dhcp/dhcpd.conf` раскомментируйте строку `authoritative`. Эта настройка нужна для того, чтобы сервер автоматически разрешал все недействительные IP-адреса в сети и назначал новые и действительные адреса каждому новому зарегистрированному устройству без вмешательства пользователя.
- Теперь нужно указать имя сетевого интерфейса в файле `/etc/default/isc-dhcp-server`, чтобы сервер знал, какое сетевое устройство использовать.

Если вы не помните имя интерфейса, запустите команду `ip addr` и найдите нужный интерфейс. В нашем случае в системе есть как Ethernet, так и беспроводные интерфейсы, и для сервера DHCP мы выберем интерфейс Ethernet, который называется `enp0s25`. Соответствующая настройка в файле `/etc/default/isc-dhcp-server` должна выглядеть так:

```
INTERFACESv4="enp0s25"
```

4. Сохраните изменения в файле и перезапустите службу DHCP:

```
sudo systemctl restart isc-dhcp-server.service
```

Теперь у вас есть рабочий сервер DHCP под управлением Linux. Сервер DHCP дает вам некоторые преимущества в управлении локальной сетью, но бывают случаи, когда вам не нужно создавать новый сервер, потому что все сетевые маршрутизаторы сами по себе предоставляют полностью рабочую службу DHCP.

Скорее всего, на маршрутизаторе в вашей локальной сети уже есть полнофункциональный сервер DHCP, и он будет конфликтовать с тем, который вы настроили. В этом случае служба `isc-dhcp-server` сообщит об ошибке, суть которой в том, что ей не удалось подключиться ни к одному интерфейсу.

Учитывая предыдущее примечание, будьте готовы столкнуться с ошибкой, когда будете проверять состояние службы DHCP с помощью стандартной команды:

```
sudo systemctl status isc-dhcp-server.service
```

В нашем случае ошибка выглядит так:

```
* isc-dhcp-server.service - ISC DHCP IPv4 server
Loaded: loaded (/lib/systemd/system/isc-dhcp-server.service; enabled; vendor preset: enabled)
Active: failed (Result: exit-code) since Thu 2023-06-22 11:33:43 UTC; 31min ago
Docs: man:dhcpd(8)
Process: 1716 ExecStart=/bin/sh -ec CONFIG_FILE=/etc/dhcp/dhcpd.conf; if [ -f /etc/ltsp/dhcpd.conf ]; then CONFIG_FILE=/etc/ltsp/dhcpd.conf; fi
Main PID: 1716 (code=exited, status=1/FAILURE)
CPU: 12ms

Jun 22 11:33:43 nuc5 dhcpd[1716]:
Jun 22 11:33:43 nuc5 dhcpd[1716]: Not configured to listen on any interfaces!
Jun 22 11:33:43 nuc5 dhcpd[1716]:
Jun 22 11:33:43 nuc5 dhcpd[1716]: If you think you have received this message due to a bug rather
Jun 22 11:33:43 nuc5 dhcpd[1716]: than a configuration issue please read the section on submitting
Jun 22 11:33:43 nuc5 dhcpd[1716]: bugs on either our web page at www.isc.org or in the README file
Jun 22 11:33:43 nuc5 dhcpd[1716]: before submitting a bug. These pages explain the proper
Jun 22 11:33:43 nuc5 dhcpd[1716]: process and the information we find helpful for debugging.
Jun 22 11:33:43 nuc5 dhcpd[1716]:
Jun 22 11:33:43 nuc5 dhcpd[1716]: exiting.
```

Рис. 13.1. Ошибка при запуске службы DHCP

Как только вы изолируете свою систему от локальной сети и начнете использовать ее в качестве единственного сервера DHCP, служба заработает как положено.

В следующем разделе мы покажем, как настроить сервер NFS в локальной сети.

Как настроить сервер NFS

NFS — это распределенная файловая система, которая обеспечивает совместный доступ к файлам по сети. Чтобы продемонстрировать, как она работает, мы установим сервер NFS на одной из наших машин с Ubuntu 22.04.2 LTS. Более подробные теоретические сведения об NFS вы найдете в главе 7 «Сетевое окружение Linux».

Файловая система NFS поддерживается в любой среде Linux и Unix, а также в Windows, но с некоторыми ограничениями. Для клиентских сред, которые работают преимущественно под управлением Windows, лучше использовать протокол SMB/**CIFS**. Кроме того, если вы беспокоитесь о конфиденциальности и безопасности, пожалуйста, имейте в виду, что протокол NFS сам по себе не шифруется, поэтому по умолчанию данные передаются в незащищенном виде.

Как установить и наладить сервер NFS

Мы будем использовать компьютер под управлением Ubuntu в качестве сервера и покажем, как получить доступ к файлам с другого клиента Linux. Сначала нужно установить и настроить сервер NFS таким образом:

1. Установите пакет `nfs-kernel-server`:

```
sudo apt install nfs-kernel-server
```
2. Запустите установленную службу с помощью `systemctl`, а затем проверьте ее состояние:

```
sudo systemctl start nfs-kernel-server
sudo systemctl enable nfs-kernel-server
sudo systemctl status nfs-kernel-server
```

3. Теперь нужно определиться с тем, какой каталог или каталоги мы будем экспортировать для доступа по NFS. Например, можно экспортировать уже существующий каталог `/home`, а можно создать специальный общий каталог. Каталоги могут располагаться в любом месте файловой системы: например, общий каталог может находиться непосредственно в корневом каталоге (`/`), но может быть и внутри таких стандартных каталогов, как `/var`, `/mnt` или `/srv`, — выбор за вами.

В этом упражнении мы создадим новый каталог `/export/shares` внутри каталога `/home`:

```
sudo mkdir -p /home/export/shares
```

4. Установите права доступа 777 к новому каталогу, потому что в нашем примере мы не будем использовать аутентификацию **LDAP**:

```
sudo chmod 777 /home/export/shares/
```

5. Теперь каталог готов к экспорту. У службы NFS есть три конфигурационных файла: `/etc/default/nfs-kernel-server`, `/etc/default/nfs-common` и `/etc/exports`, но нам понадобится отредактировать только последний из них.

Откройте файл `/etc/exports` в текстовом редакторе. Возможно, вы обратили внимание, что внутри файла уже есть закомментированные примеры двух типов директив: один пример для NFS версий 2 и 3, а другой — для версии 4. Чтобы подробнее узнать, чем они различаются, посмотрите эту презентацию: https://archive.fosdem.org/2018/schedule/event/nfs3_to_nfs4/attachments/slides/2702/export/events/attachments/nfs3_to_nfs4/slides/2702/FOSDEM_Presentation_Final_pdf.pdf.

6. Теперь давайте отредактируем файл `/etc/exports`. Для каждого расположения в нем нужна строка такого вида:

/путь/к/файлам IP(параметры)

В нашем упражнении всего один общий каталог `/home/export/shares/`, поэтому мы добавим одну строку. Если вы хотите предоставить доступ к этому каталогу всем клиентам в сети, можете указать класс подсети (например, `192.168.0.0/24`). Но если доступ к общим ресурсам нужен только определенным клиентам, перечислите их IP-адреса в одной строке через пробел. В конфигурации NFS существует множество параметров, с которыми можно ознакомиться на справочной странице `man exports`. В нашем случае давайте добавим такие параметры:

<code>rw</code>	Предоставить доступ для чтения и записи
<code>sync</code>	Подтверждать запись изменений на диск (надежнее, но медленнее)

В итоге мы добавим в конфигурационный файл `/etc/exports` такую строку. Обратите внимание, что в ней должен быть всего один пробел — между каталогом и IP-адресом:

`/home/export/shares 192.168.0.0/24(rw,sync)`

7. После того как вы сохраните и закроете файл, перезапустите службу NFS:

```
sudo systemctl restart nfs-kernel-server
```

8. Затем примените обновленную конфигурацию:

```
sudo exportfs -a
```

Ключ `-a` нужен для того, чтобы экспортировать сразу все каталоги, которые перечислены в конфигурационном файле.

9. После перезапуска службы стоит разрешить доступ к NFS в межсетевом экране. Служба NFS по умолчанию использует порт **2049**. Поскольку мы разрешаем доступ к общим ресурсам для всех систем из нашей сети, мы добавим такое правило:

```
sudo ufw allow nfs
```

10. Проверьте, что правило успешно добавлено и порт 2049 открыт для всех клиентов:

```
packt@nuc5:~$ sudo ufw status
Status: active
```

To	Action	From
--	-----	----
2049	ALLOW	Anywhere
2049 (v6)	ALLOW	Anywhere (v6)

11. Если ваш межсетевой экран отключен, его можно включить такой командой:

```
sudo ufw enable
```

Итак, мы настроили сервер NFS. Чтобы получить к нему доступ с другой системы, на ней нужно развернуть клиент NFS. В следующем разделе мы покажем, как это сделать.

Как настроить клиент NFS

В качестве клиента NFS мы будем использовать другой компьютер — ноутбук под управлением Debian GNU/Linux 12. Сначала на нем нужно установить клиентское ПО с помощью такой команды:

```
sudo apt install nfs-common
```

После этого создайте на этом же компьютере каталог, в который будут монтироваться общие ресурсы сервера NFS. Мы создадим каталог `/home/shares` таким образом:

```
sudo mkdir /home/shares
```

Теперь, когда новый каталог создан, в него можно смонтировать общий каталог на сервере:

```
sudo mount 192.168.0.113:/home/export/shares /home/shares
```

Обратите внимание, что в качестве первого аргумента команды `mount` мы указали расположение на сервере, а в качестве второго — на клиенте. С помощью команды `df -h` можно убедиться, что монтирование прошло успешно: новая точка монтирования отображается последней в выводе команды:

```
alexandru@debian:~$ df -h
Filesystem                                Size Used Avail Use% Mounted on
udev                                      12G   0  12G   0% /dev
# ... другие файловые системы ...
/dev/sdal                                115G  50G   65G  44% /media/alexandru/Ventoy
192.168.0.113:/home/export/shares         98G  7.8G   86G   9% /home/shares
```

Вот мы и закончили настраивать общие ресурсы NFS. Теперь нужно протестировать конфигурацию, чтобы убедиться, что она работает.

Как протестировать конфигурацию NFS

После того как мы настроили сервер и клиент NFS, стоит проверить, все ли работает нормально. Для этого мы создали несколько файлов с именами типа `testing_filesX` на сервере (Ubuntu) в каталоге `/home/export/shares` и еще несколько файлов с подобными именами на клиенте (Debian) в каталоге `/home/shares`. И те и другие файлы отображаются в соответствующих каталогах на обоих компьютерах, и с ними можно выполнять обычные файловые операции:

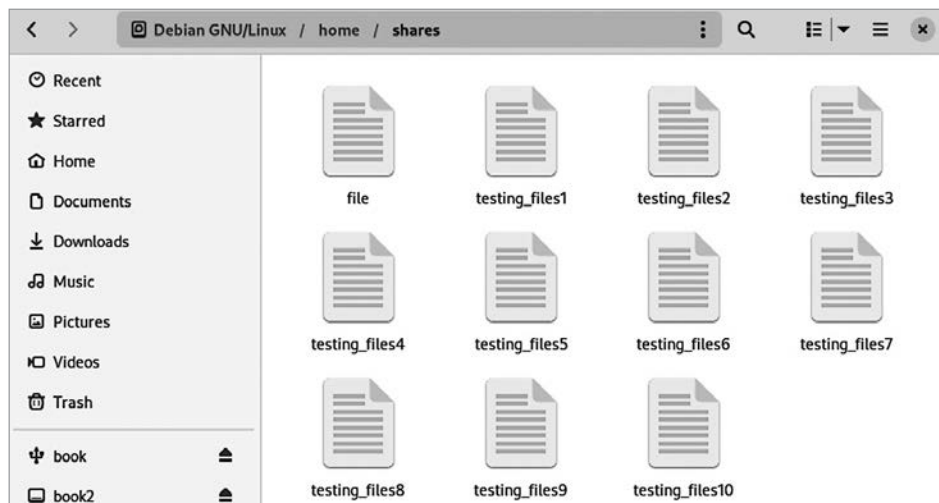


Рис. 13.2. Тестирование NFS на локальном клиенте

Итак, и сервер и клиент NFS функционируют нормально, и к общим ресурсам NFS можно обращаться и через **графический интерфейс** (как показано на предыдущем снимке экрана), и через **командную строку**.

В следующем разделе вы узнаете, как настроить общий ресурс Samba/CIFS, к которому смогут получить доступ клиенты Windows в сети.

Как настроить файловый сервер Samba

Файловый сервер Samba позволяет наладить общий доступ к файлам в сети, где клиенты используют разные операционные системы, например Windows, macOS и Linux. В этом разделе мы настроим сервер Samba на Ubuntu 22.04.2 LTS и обеспечим доступ к общим ресурсам с узлов, на которых установлены разные операционные системы. Протокол SMB/CIFS разработала компания Microsoft; узнать о нем подробнее можно на странице learn.microsoft.com/en-us/windows/win32/fileio/microsoft-smb-protocol-and-cifs-protocol-overview. Некоторые сведения об этом протоколе также приводились в главе 7 «Сетевое окружение Linux». В сле-

дующих подразделах мы покажем, как установить и настроить сервер Samba в локальной сети.

Как установить и наладить сервер Samba

Установка состоит из таких шагов:

1. Сначала установите Samba на систему Linux:

```
sudo apt install samba
```

2. После установки проверьте, работает ли служба Samba:

```
sudo systemctl status smbd
```

3. Если все в порядке, создайте каталог, к которому мы настроим общий доступ по сети. Для примера создадим его внутри нашего домашнего каталога:

```
mkdir /home/packt/samba_shares
```

4. Теперь давайте отредактируем конфигурационный файл Samba `/etc/samba/smb.conf`. Этот файл состоит из нескольких разделов, имена которых заключены в квадратные скобки. Например, в разделе `[global]` содержатся общие настройки службы, а каждый общий ресурс конфигурируется в собственном разделе (например, мы назовем такой раздел `[shares]`). Из соображений безопасности имеет смысл создать резервную копию исходного конфигурационного файла, прежде чем его редактировать. Это можно сделать так:

```
sudo mv /etc/samba/smb.conf /etc/samba/smb.conf.original
```

5. С помощью предыдущей команды мы переименовали исходный файл конфигурации, так что давайте теперь создадим новый файл с тем же именем `smb.conf`, а внутри этого файла — раздел `[global]` с такими директивами:

```
[global]
server string = Local Samba File Server
# Отображаемое имя сервера Samba
workgroup = WORKGROUP
# Рабочая группа
map to guest = Bad User
# Пользователи с неизвестными именами аутентифицируются
# под гостевой учетной записью
usershare allow guests = yes
# Общие ресурсы доступны пользователям без аутентификации
interfaces = lo eno1
# Сетевые интерфейсы, через которые работает Samba
# (в нашем случае lo — обратная петля, eno1 — Ethernet);
# уточните имена своих интерфейсов с помощью ip addr
# или ip link
bind interfaces only = yes
# Какие сетевые интерфейсы могут обслуживать запросы SMB;
```

```

# подробнее см. man smb.conf
server role = standalone server
# Доступ по аутентификации
# (в том числе под гостевой учетной записью)
hosts allow = 192.168.0.0/16
# Узлы, с которых разрешен доступ
name resolve order = bcast host
# Порядок разрешения имен: сначала опрашивать сетевые интерфейсы,
# затем стандартные службы DNS

```

6. Теперь создайте раздел `[shares]` с необходимыми параметрами общего каталога:

```

[shares]
comment = Samba shares
# Отображаемое имя общего каталога
path = /home/packt/samba_shares
create mask = 0664
force create mode = 0664
directory mask = 0775
force directory mode = 0775
# Настройки прав доступа для обмена файлами с другими ОС
writable = yes
read only = no
guest ok = yes
browsable = yes
valid users = @packt @alex
# Пользователи, которым разрешен доступ

```

7. Сохраните конфигурационный файл и перезапустите службу Samba:

```
sudo systemctl restart smbd
```

8. Не забудьте разрешить доступ к Samba в межсетевом экране:

```
sudo ufw allow samba
```

9. Наконец, можно запустить команду `testparm`, чтобы убедиться, что в конфигурационном файле `smb.conf` нет синтаксических ошибок:

```

packt@nuc7:~$ testparm
Load smb config files from /etc/samba/smb.conf
Loaded services file OK.
Weak crypto is allowed by GnuTLS (e.g. NTLM as a compatibility fallback)

```

```
Server role: ROLE_STANDALONE
```

```
Press enter to see a dump of your service definitions
```

Если вы видите `Loaded services file OK`, это значит, что конфигурация синтаксически корректна.

Теперь можно настроить учетные записи, которые смогут обращаться к общим ресурсам Samba. В следующем разделе мы создадим новых пользователей и группы Samba.

Как создавать учетные записи Samba

На каждом сервере Samba должны быть настроены учетные записи пользователей, у которых есть доступ к общим файлам и каталогам. Эти пользователи должны быть одновременно пользователями Samba и пользователями системы, потому что иначе они не смогут аутентифицироваться и обращаться к системным файлам. Предположим, что вам нужно создать локальный общий каталог для вашей небольшой компании или членов семьи.

Если вы создаете локальные учетные записи специально для доступа к общим ресурсам Samba, вам не требуется, чтобы они вели себя как настоящие пользователи, потому что им нужен только доступ к общим ресурсам. Однако локальные пользователи Samba должны быть также локальными пользователями системы. В нашем случае мы создадим новую учетную запись Samba, которая соответствует нашей обычной учетной записи `packt`:

1. Чтобы добавить локального пользователя `packt` в Samba, запустите такую команду:

```
sudo smbpasswd -a packt
```

2. Поскольку мы выполняли предыдущие операции от имени этого же пользователя `packt`, он также владеет каталогом `samba_shares`, который мы создали в начале этого раздела. Владельца можно «обнулить» с помощью такой команды:

```
sudo chown -R nobody:nogroup /home/packt/samba_shares
```

3. А что, если нужно использовать другую учетную запись? Допустим, мы добавили в систему нового пользователя `alex`:

```
sudo adduser alex
```

4. Соответствующую учетную запись Samba можно создать так же, как предыдущую:

```
sudo smbpasswd -a alex
```

5. Обратите внимание, что в файле `smb.conf` мы уже предусмотрели доступ для указанных пользователей:

```
valid users = @packt @alex
```

Перед тем как проверить доступ пользователя `alex` к Samba, нужно предоставить ему доступ к общему каталогу. Это можно сделать так:

6. Установите пакет `acl`:

```
sudo apt install acl
```

7. С помощью команды `setfacl` предоставьте учетной записи `alex` права на чтение, запись и выполнение для каталога `/home/packt/samba_shares`:

```
sudo setfacl -R -m "u:alex:rwX" /home/packt/samba_shares
```


В следующем разделе мы покажем, как получить доступ к общим ресурсам Samba с различных систем в сети.

Как получить доступ к общим ресурсам Samba

В вашей сети теперь можно обратиться к общим ресурсам Samba из Linux, macOS или Windows. Давайте для примера посмотрим, как получить доступ к этим ресурсам в Linux, используя только командную строку. (Доступ из графического интерфейса или с помощью клиента Windows или macOS мы оставляем вам в качестве самостоятельного упражнения.)

Сначала нужно установить клиент Samba `smbclient`; это делается примерно одинаково во всех основных дистрибутивах Linux. Например, в Ubuntu/Debian клиент Samba можно установить так (предварительно убедитесь, что ваши репозитории обновлены):

```
sudo apt install smbclient
```

Если вы работаете с Fedora/RHEL, запустите такую команду:

```
sudo dnf install samba-client
```

Давайте теперь запросим общие ресурсы пользователя `packt` с одной из наших локальных машин. Помните, что общий каталог Samba находится на локальном сервере под управлением Ubuntu. На клиентском компьютере запустите такую команду, подставив IP-адрес своего сервера Samba:

```
alexandru@debian:~$ smbclient //192.168.0.140/shares -u packt
Password for [WORKGROUP\packt]:
Try "help" to get a list of possible commands.
smb: \> ls
.                D    0   Sat Jul   8 09:56:43 2023
..               D    0   Fri Jul   7 11:42:44 2023
new_file         N    0   Thu Jan   1 02:00:00 1970
                102626232 blocks of size 1024. 72091436 blocks available
```

В этом листинге видно, что доступ к Samba успешно предоставлен, и пользователю `packt` видны общие ресурсы на сервере Samba. Чтобы задать имя пользователя для аутентификации, мы использовали ключ `-U`. В качестве местоположения мы указали IP-адрес сервера, а затем имя общего ресурса (`shares`). Если нам понадобится узнать, какие службы Samba доступны на сервере, можно запустить команду `smbclient` с ключом `-L`:

```
alexandru@debian:~$ smbclient -L //192.168.0.140 -U packt
Password for [WORKGROUP\packt]:
      Sharename      Type      Comment
      -----
      shares         Disk      Samba shares
      IPC$           IPC       IPC Service (Local Samba File Server)
```

Вот мы и подошли к концу главы о том, как настраивать серверы Linux. Мы рассмотрели серверы, которые считаем особенно важными и актуальными для

любого системного администратора Linux. Однако в Linux бывает и множество других серверов, которые нам не удалось охватить из-за ограничений по объему книги. Впрочем, в интернете можно найти немало дополнительных ресурсов; например, можно начать с официальной документации по RHEL, Ubuntu или Debian, где рассматривается большинство типов серверов Linux. Один из этих типов — веб-сервер, и вам не помешает знать, как его настраивать. Смело находите и изучайте и любые другие ресурсы, которые кажутся вам важными, даже если они не включены в список дополнительной литературы.

Итоги

В этой главе вы узнали, как устанавливать и настраивать наиболее известные службы, которые доступны для Linux. Всякому администратору Linux важно уметь конфигурировать различные серверы — от DNS до DHCP, Apache и сервера печати.

Изучив эту главу, вы научились налаживать многие серверы Linux. Вы узнали, как настроить SSH, как обмениваться файлами с помощью протоколов NFS и Samba/CIFS, а также как настроить DNS и локальный сервер DHCP. Одним словом, из этой главы вы почерпнули много нового, и все же мы охватили лишь малую часть того, что относится к администрированию серверов Linux.

В следующей главе вы познакомитесь с облачными технологиями.

Вопросы

Теперь, когда вы хорошо представляете себе, как управлять некоторыми из наиболее популярных служб в Linux, предлагаем вам несколько упражнений для того, чтобы отточить свои навыки:

1. Попробуйте воспроизвести все упражнения этой главы не в локальной сети, а на VPS.
2. Попробуйте настроить в Ubuntu стек Linux — nginx — PostgreSQL — Python.
3. Протестируйте все службы, о которых шла речь в этой главе, в дистрибутиве на базе Fedora или RHEL.

Дополнительная литература

Чтобы получить дополнительные сведения обо всем, что рассматривалось в этой главе, обратитесь к официальной документации:

- Ubuntu documentation.ubuntu.com/server
- RHEL docs.redhat.com/en/documentation/red_hat_enterprise_linux
- nginx docs.nginx.com/nginx

Часть 4

ОБЛАЧНОЕ АДМИНИСТРИРОВАНИЕ

В четвертой части этой книги вы познакомитесь с новейшими понятиями из сферы облачных вычислений. Вы научитесь использовать специальные инструменты, которые позволяют автоматизировать настройку и развертывание приложений, такие как Kubernetes и Ansible, а также развертывать Linux в облачных службах AWS и Azure.

Эта часть состоит из четырех глав:

- *Глава 14. Краткое введение в облачные вычисления*
- *Глава 15. Как развертывать Linux в облаке с помощью AWS и Azure*
- *Глава 16. Как развертывать приложения с помощью Kubernetes*
- *Глава 17. Как налаживать инфраструктуру и автоматизацию с помощью Ansible*

14

Краткое введение в облачные вычисления

В этой главе вы познакомитесь с основами **облачных вычислений** и фундаментальными понятиями облачных инфраструктур. Вы узнаете о решениях класса «X как услуга», таких как **инфраструктура как услуга (IaaS)**, **платформа как услуга (PaaS)**, **программное обеспечение как услуга (SaaS)** и **контейнеры как услуга (CaaS)**. Мы поговорим об основных стандартах облачных технологий, методологии **DevOps**, **CI/CD** и **микросервисах**. Вы получите начальное представление об AWS, Azure и других облачных решениях. К концу этой главы мы также познакомим вас с такими инструментами, как **Ansible** и **Kubernetes**. Эта глава служит кратким теоретическим введением, и ее цель — сформировать основу для следующих трех глав, которые посвящены облачным технологиям и из которых вы получите практические навыки работы со всеми этими продуктами.

В этой главе мы рассмотрим такие темы:

- Введение в облачные технологии.
- Решения класса «X как услуга»: IaaS, PaaS, CaaS.
- Введение в DevOps.
- Инструменты для управления облачными решениями.

Технические требования

Поскольку эта глава носит чисто теоретический характер, в ней нет никаких специальных технических требований. Все, что вам понадобится, — это желание узнать об облачных технологиях.

Введение в облачные технологии

Термин «облачные вычисления», или просто «облако», занял прочное место в лексиконе как профессиональных специалистов, так и многочисленных энтузиастов в области **ИТ**. Не обязательно даже иметь отношение к компьютерным технологиям, чтобы довольно часто слышать и говорить о чем-нибудь «облач-

ном». Современная индустрия высоких технологий стремительно меняется, и один из краеугольных камней этих изменений — облачные решения. Исследователи считают, что термин «облачные вычисления» впервые был использован в 1996 году в бизнес-плане компании Compaq (см. technologyreview.com/2011/10/31/257406/who-coined-cloud-computing/).

Облачные вычисления — относительно старая идея, хотя такое название она получила лишь недавно. Эта модель вычислений использовалась уже в самых первых компьютерах. Например, в 1950-х годах вычислительные машины в основном были мейнфреймами, доступ к которым осуществлялся с различных терминалов. Эта модель похожа на современные облачные решения, когда службы размещаются на мощностях облачных провайдеров и предоставляются через интернет пользователям различных терминалов — от настольных компьютеров до смартфонов, планшетов и ноутбуков. В основе этой модели лежат чрезвычайно сложные технологии, ориентироваться в которых необходимо каждому современному системному администратору.

Возможно, вы задаётесь вопросом, почему в книге по администрированию Linux есть целый крупный блок про облачные вычисления и облачные технологии. Это связано с тем, что за последние десять лет Linux занял центральное место в облачных решениях точно так же, как до этого завоевал интернет и отрасль высокопроизводительных вычислений. По данным исследовательского проекта TOP500, уже много лет подряд 500 ведущих суперкомпьютеров мира работают под управлением Linux (см. top500.org/lists/top500). Чтобы облако функционировало, ему нужна операционная система; и хотя это не обязательно должна быть Linux, именно эта ОС используется на большинстве публичных облачных серверов (см. redhat.com/rhdc/managed-files/cl-state-of-linux-in-public-cloud-for-enterprises-f11154kc-201802-en_0.pdf). Это во многом объясняется тем, что открытый исходный код Linux оказывается важным фактором для технических специалистов как в государственном, так и в частном секторе.

В следующем разделе мы поговорим об облачных стандартах и о том, почему о них полезно иметь представление, если вы планируете развертывать или сопровождать облачный экземпляр.

Стандарты облачных вычислений

Прежде чем предметно разговаривать об облачных вычислениях, давайте вкратце познакомимся с соответствующими стандартами и выясним, какую роль они играют в современных облачных средах. Вы наверняка знаете, что почти каждый аспект компьютерных технологий регулируется какими-нибудь стандартами или регламентами.

Облачные вычисления — не исключение из этого правила, и вас может удивить, как много ассоциаций, комитетов и регулирующих органов участвуют в раз-

работке соответствующих стандартов. Эта книга не ставит перед собой задачу охватить все эти организации и стандарты, но в следующих разделах мы опишем те из них, которые кажутся нам наиболее важными и актуальными для современных облачных служб и веб-приложений.

Стандарты ISO и IEC

Две самые известные организации по стандартизации — **Международная организация по стандартизации (ISO)** и **Международная электротехническая комиссия (IEC)**, которые поддерживают около 30 опубликованных и разрабатываемых стандартов на облачные вычисления и распределенные платформы. У этих организаций есть совместная рабочая группа по стандартизации базовой инфраструктуры облачных вычислений, платформ потребительских приложений и служб — технический подкомитет **ISO/IEC JTC 1/SC 38**.

Вот некоторые семейства облачных стандартов ISO/IEC:

Соглашения об уровне обслуживания (SLA)	ISO/IEC 19086
Сервис-ориентированная архитектура (SOA)	ISO/IEC TR 30102 ISO/IEC 18384
OVF (Open Virtualization Format)	ISO/IEC 17203
Соглашения о совместном использовании данных (DSA)	ISO/IEC CD 23751

Ознакомиться с фрагментами этих стандартов и приобрести их можно на странице iso.org/committee/601355/x/catalogue.

Инициатива по координации облачных стандартов (CSC)

Еще одна структура, которая разрабатывает стандарты облачных технологий, — Координационный совет по облачным стандартам (**CSC**), созданный **Европейской комиссией** совместно со специализированными организациями.

В 2012 году ЕК совместно с **Европейским институтом телекоммуникационных стандартов (ETSI)** запустила CSC для того, чтобы разрабатывать стандарты и политики безопасности, совместимости и переносимости облачных вычислений. Эта инициатива состояла из двух этапов; второй из них начался в 2015 году и завершился финальными отчетами (версия 2.1.1):

Потребности пользователей облачных вычислений	ETSI SR 003 381
Стандарты облачных вычислений и открытый исходный код	ETSI SR 003 382
Совместимость и безопасность облачных вычислений	ETSI SR 003 391
Оценка зрелости стандартов облачных вычислений	ETSI SR 003 392

Найти эти и другие стандарты ETSI можно на сайте etsi.org/standards.

Национальный институт стандартов и технологий США

Наш список продолжает одна из самых известных организаций по стандартизации, о которой уже не первый раз упоминается в этой книге, — **Национальный институт стандартов и технологий (NIST)** при Министерстве торговли США. Основная задача NIST — стандартизировать безопасность и совместимость для государственных учреждений США, поэтому всем, кому интересно сотрудничать с этими учреждениями, стоит ознакомиться с документацией NIST по облачным технологиям. Соответствующие стандарты доступны по адресу nist.primo.exlibrisgroup.com/discovery/collectionDiscovery?vid=01NIST_INST:01NIST&collectionId=8126389620008106&query=any,contains,cloud.

Международный союз электросвязи

Мы завершаем наш список одним из старейших, если не самым старым органом по разработке стандартов, — **Международным союзом электросвязи (МСЭ)**, который был основан в 1865 году и сейчас является специализированным учреждением ООН. Его основная задача — разрабатывать стандарты для электросвязи и, в частности, регулировать международное распределение радиочастот и спутниковых орбит. В ведении МСЭ также находится азбука Морзе как стандартный механизм связи. МСЭ предлагает множество стандартов и рекомендаций, которые касаются глобальной информационной инфраструктуры, протоколов интернета, сетей нового поколения, **интернета вещей** и «умных» городов. С этими документами можно ознакомиться по адресу itu.int/ITU-T/recommendations/index.aspx?ser=Y, а чтобы сузить список до стандартов по облачным вычислениям, разверните ветвь Y.3500-Y.3599: Cloud Computing. Стандартами облачных вычислений занимается **исследовательская группа SG13** в рамках проекта **JCA-Cloud**.

Есть и немало других организаций, которые разрабатывают стандарты облачных технологий, в том числе:

CSCC	Совет заказчиков облачных стандартов	Ныне не функционирует
DMTF	Целевая группа по распределенному управлению	dmtf.org/standards/cloud
OASIS	Организация по продвижению стандартов по структурированной информации	oasis-open.org/standards

Стандарты принимаются прежде всего затем, чтобы и **облачным провайдерам**, и клиентам было удобнее работать с облачными службами. И тем и другим нужен легкий доступ к данным, который особенно важен для провайдеров и разработчиков приложений, потому что от него зависит оперативность и совместимость. Однако стандарты должны быть не только технически грамотными, но и последовательными, а также устойчивыми. Фактически большинство стандартов делится на две основные категории: те, которые сформировались на основе практики, и те, которые разрабатывались специально. Важный тип стандартов второй категории — **интерфейсы прикладного программирования (API)**. Стандартиза-

ция фреймворков приложений, сетевых протоколов и API — существенный фактор успеха для всех, кто работает в соответствующей индустрии.

Облачные технологии с точки зрения стандартов API

API — это наборы протоколов, процедур и функций, то есть всех элементов, из которых складывается распределенное веб-приложение. Современные облачные API появились в начале XXI века, и многие из них восходят к магистерской диссертации Роя Филдинга¹. Предшественниками нынешних API были стандарты сервис-ориентированной архитектуры (SOA) и **протокол SOAP**, который опирался на **XML**. Современные API основаны на новом архитектурном стиле приложений, который называется **REST**².

API в стиле REST опираются на архитектурные шаблоны, элементы, связки и представления, которые описал Филдинг в своей диссертации. Для того чтобы API относился к REST, он должен удовлетворять шести основным требованиям:

- унифицированные пользовательские интерфейсы;
- четкое разграничение между клиентом и сервером;
- операции без сохранения состояния;
- кэшируемые ресурсы;
- многоуровневая структура серверов;
- выполнение кода по требованию.

Эти руководящие принципы сами по себе нельзя считать стандартом; в идеологии REST они предоставляются разработчикам как высокоуровневый слой абстракции. Если бы они не были стандартизированы, то так и оставались бы умозрительными принципами, которые хороши в теории, но на практике порождают путаницу и разочарование среди разработчиков.

Единственная организация, которой удалось стандартизировать REST API для облачных вычислений, — это DMTF: она разработала модель **CIMI** (Cloud Infrastructure Management Interface, интерфейс управления облачной инфра-

¹ Roy Thomas Fielding. Architectural Styles and the Design of Network-based Software Architectures (Архитектурные стили и проектирование сетевых программных архитектур). Калифорнийский университет в Ирвайне, 2000 (roy.gbiv.com/pubs/dissertation/top.htm). — *Примеч. ред.*

² Сокр. Representational State Transfer (передача репрезентативного (самоописываемого) состояния). В английском языке аббревиатуры (в том числе в области IT) нередко составляются так, чтобы образовать значащее слово (в данном случае *rest* — многозначное слово, среди значений которого, например, *отдых* и *остаток*, а соответствующее прилагательное RESTful буквально означает *успокоительный*). Ради этого эффекта авторы аббревиатур часто пренебрегают семантикой, в результате чего буквальный смысл названия может не в полной мере отражать суть соответствующего понятия или даже вводить в заблуждение. — *Примеч. пер.*

структурой) и REST-версию протокола **HTTP** (стандарт DSP0263, который доступен на странице dmtf.org/dsp/DSP0263).

Существуют и другие спецификации, претендующие на статус возможных будущих стандартов, на основе которых разработчики смогут проектировать REST API. Среди них — спецификация **OpenAPI**: это промышленный стандарт, который описывает разработку API независимо от языка (см. spec.openapis.org/oas/latest.html). Отметим также **GraphQL** — язык запросов и серверную среду выполнения, для которой существуют реализации на многих языках, включая Python, JavaScript, Scala, Ruby и PHP.

Разработчики предпочитают REST API, потому что он понятный и легковесный, а код на его основе просто писать и удобно сопровождать. Он эффективнее, чем конкурирующие стандарты, порождает меньше трафика и поддерживает множество форматов данных, из которых предпочтительным является **JSON**. Данные в этом формате легко читать и записывать, и он обеспечивает хорошую совместимость между приложениями на разных языках — например, JavaScript, Ruby, Python и Java. Практика показала, что JSON в качестве формата данных по умолчанию для API удобен, хорошо масштабируется и не зависит от платформы.

API повсеместно распространены в интернете и в облаке и служат основой для сервис-ориентированной архитектуры и микросервисов. Например, в микросервисах REST API позволяет службам взаимодействовать между собой, предлагая оптимизированную архитектуру для облачных распределенных ресурсов.

Таким образом, если вы хотите овладеть облачными технологиями, вам стоит ознакомиться с соответствующими стандартами. В следующем разделе мы обсудим типы и архитектуру облачных решений.

Архитектура облачных решений

Архитектура облака похожа на архитектуру здания. Проектировать ее начинают с чистого листа, на котором архитекторы соединяют различные стандартизированные компоненты, чтобы получить архитектурный проект. Конечный результат придерживается определенного архитектурного стиля. Облачную архитектуру разрабатывают по такому же принципу.

Облако основано на сетевом клиент-серверном архитектурном стиле с многоуровневой структурой без сохранения состояния. Его базовые компоненты — REST API, SOA, микросервисы и веб-технологии. Эталонная архитектура облака определена в стандарте NIST 500-292 (nist.gov/publications/nist-cloud-computing-reference-architecture).

Некоторые из технологий, на которые опираются облачные вычисления, уже обсуждались в главе 11 «Как работать с виртуальными машинами» и в главе 12 «Как управлять контейнерами с помощью Docker». В самом деле: и виртуализация, и контейнеры — это фундаментальные технологии облачных вычислений.

Представьте себе ситуацию, когда вам нужны несколько систем под управлением Linux для того, чтобы разворачивать свои приложения. Сначала вы обращаетесь к облачному провайдеру и запрашиваете нужные системы. Провайдер на своей инфраструктуре создает **виртуальные машины** в соответствии с вашими требованиями, помещает их все в одну сеть и предоставляет вам учетные данные для того, чтобы обращаться к ним. В результате вы получаете доступ к системам и вносите за это абонентскую плату — либо повремененно (например, ежедневно, ежемесячно или ежегодно), либо в зависимости от потребления ресурсов. Чаще всего запросы к провайдеру передаются через специальный веб-интерфейс, который разработан под нужды пользователей.

Все технологии, на которых строятся облачные решения, опираются на виртуальные машины и контейнеры. Внутри облака все абстрагировано и автоматизировано. В следующем разделе мы расскажем, какие типы служб доступны в облаке.

Типы облачной инфраструктуры и облачных служб

К какому бы типу ни относилось облако, у него есть определенная архитектура, о которой шла речь в предыдущем разделе, — это своего рода шаблон, на который опираются облачные вычисления. Облачная архитектура служит основой для облачной инфраструктуры, а инфраструктура — основой для облачных служб. Видите, как все взаимосвязано? Теперь давайте посмотрим, что представляют собой инфраструктура и службы применительно к облаку.

Существует четыре основных типа **облачной инфраструктуры**:

- **Публичные облака** функционируют на инфраструктуре, которая принадлежит провайдеру и, как правило, размещена отдельно от собственной инфраструктуры клиентов. Крупнейшие провайдеры публичных облаков — AWS, Microsoft Azure и Google Cloud.
- **Частные облака** предназначены для отдельных пользователей и групп с изолированным доступом; их можно размещать как в локальной, так и в удаленной инфраструктуре. Частным облаком может управлять сама организация, которая его эксплуатирует, или внешний провайдер.
- **Гибридные облака** объединяют частные и публичные облака, которые функционируют внутри связующей среды, где ресурсы потенциально можно масштабировать по требованию.
- **Мультиоблака** объединяют несколько облачных решений от нескольких провайдеров.

Помимо типов облачной инфраструктуры, есть также четыре основных вида **облачных служб**:

- **IaaS (инфраструктура как услуга)**. Облачный провайдер управляет всей аппаратной инфраструктурой, такой как серверы и сетевые компоненты, а также виртуализацией и хранением данных. Провайдер владеет инфраструктурой, а пользователь ее арендует и управляет операционной системой,

средами выполнения, автоматизацией, системами контроля и контейнерами, а также данными и приложениями. IaaS — это основа любой службы облачных вычислений, потому что она предоставляет базовые ресурсы.

- **СаaS (контейнеризация как услуга)** считается подмножеством IaaS и характеризуется теми же преимуществами, только опирается не на виртуальные машины, а на контейнеры и лучше подходит для того, чтобы развертывать распределенные системы и микросервисные архитектуры.
- **РaaS (платформа как услуга)**. Облачный провайдер управляет инфраструктурой, сетевыми компонентами и программной платформой, а пользователь владеет и управляет данными и приложениями.
- **SaaS (программное обеспечение как услуга)**. Провайдер владеет и управляет аппаратным обеспечением, сетевыми компонентами, программной платформой, системами контроля и программными приложениями. Клиенты чаще всего пользуются этой службой с помощью мобильных или веб-приложений.

Стоит упомянуть о службах еще одного вида: **бессерверные вычисления**. Несмотря на название, в этих службах задействуются серверы, но пользователи (как правило, это разработчики) не видят инфраструктуру, на которой они работают. Бессерверные вычисления также можно назвать **функцией как услугой (FaaS)**, и они позволяют выполнять модульный код по требованию, благодаря чему разработчики могут обновлять свой код в «живом» режиме. Примеры служб такого рода — AWS Lambda, Azure Functions и Google Cloud Functions. Это решение похоже на SaaS; фактически оно находится между PaaS и SaaS. В нем не нужно управлять инфраструктурой, оно хорошо масштабируется и позволяет разработчикам быстрее выводить свои приложения на рынок, а также эффективнее использует ресурсы.

Теперь, когда вы знаете, какая бывает облачная инфраструктура и каковы соответствующие службы, вы можете задаться вопросом: почему вам, вашему бизнесу или вашим знакомым стоит переходить на облачные решения? Прежде всего облачные вычисления подразумевают доступ по требованию к различным ресурсам, которыми владеет и управляет провайдер, а пользователь получает доступ к ним за абонентскую плату. Нужно ли вам переходить в облако? В следующем разделе мы обсудим преимущества и недостатки такого перехода.

Ключевые особенности облачных вычислений

Прежде чем решить, стоит ли переходить в облако, имеет смысл узнать о преимуществах и недостатках облачных вычислений. К преимуществам относятся, например, такие факторы:

- **Экономия средств**. Снижаются затраты на наладживание инфраструктуры, которой теперь управляет облачный провайдер. Пользователи могут сосредоточиться на том, чтобы разрабатывать приложения и вести бизнес.

- **Скорость, оперативность и доступ к ресурсам.** Все ресурсы облачной службы доступны за пару кликов из любого места и в любое время (впрочем, это зависит от того, насколько быстрый и надежный у вас интернет).
- **Надежность.** Ресурсы физически размещаются в разных местах, и облачный провайдер обеспечивает надлежащий контроль качества, политику **аварийного восстановления** и управление рисками, а также занимается техническим обслуживанием инфраструктуры, так что конечным пользователям не нужно тратить на это время и деньги.

Впрочем, у облачных решений есть и характерные недостатки, например:

- **Переменная производительность.** В зависимости от провайдера производительность облачных услуг может колебаться в тех или иных пределах. Впрочем, у крупных компаний (таких, как AWS, Azure или GCP) не бывает существенных проблем с производительностью; она гораздо больше зависит от скорости и качества подключения к интернету на стороне пользователя. Однако у небольших провайдеров ситуация может отличаться.
- **Время простоя** может привести к проблемам, хотя все крупные провайдеры стремятся обеспечить не менее 99,9 % времени бесперебойной работы. В случае аварии служба восстанавливается за считанные минуты, на худой конец — за несколько часов.
- **Недостаточная предсказуемость.** Хотя можно уверенно утверждать, что ни один из крупных игроков в обозримом будущем не уйдет с рынка облачных услуг, в секторе средних и малых провайдеров сохраняется неопределенность.

Мы не считаем эти факторы серьезной¹ преградой для тех, кто хочет перейти на облачные технологии. В следующем разделе мы познакомим вас с некоторыми решениями IaaS.

IaaS: инфраструктура как услуга

IaaS — основа облачных вычислений. Эта модель предлагает доступ по требованию к таким ресурсам, как вычислительные мощности, хранилища, сетевые службы и т. д. Облачные провайдеры предоставляют решения IaaS с помощью гипервизоров. В этом разделе пойдет речь о самых распространенных из этих решений. Мы подробно расскажем о таких крупных провайдерах, как **Amazon EC2** и **Microsoft Azure**, а также об альтернативных вариантах — **DigitalOcean** и **OpenStack**.

¹ Все-таки следует остерегаться от импульсивных попыток перевести все бизнес-процессы в облака. Прежде чем осуществлять переход, просчитайте финансовые трудозатраты в долгосрочной перспективе. — *Примеч. ред.*

Amazon EC2

AWS (Amazon Web Services) предоставляет решение IaaS, которое называется Amazon EC2 (Elastic Compute Cloud, «эластичное вычислительное облако»). Эта служба подходит для широкого круга пользователей, предлагая как недорогие вычислительные экземпляры, так и мощные графические процессоры для машинного обучения. AWS стала первым поставщиком IaaS во второй половине 2000-х, и на момент написания этой книги ее дела идут лучше, чем когда-либо, — даже после пандемии COVID-19 (см. statista.com/chart/18819/world-market-share-of-leading-cloud-infrastructure-service-providers).

Чтобы приступить к работе с Amazon EC2, нужно выполнить несколько шагов:

- Сначала понадобится выбрать предварительно настроенный **образ виртуальной машины (AMI)** под управлением Linux или Windows. В первом случае EC2 предлагает собственную операционную систему Amazon Linux (которая в основном двоично совместима с **RHEL**), а также большинство популярных дистрибутивов, в том числе RHEL, Ubuntu, SUSE Linux Enterprise Server и CentOS.
- Затем нужно выбрать **тип экземпляра** из очень широкого ассортимента с разными характеристиками процессоров, памяти, хранилища и сетевой инфраструктуры. Подробнее узнать о типах экземпляров EC2 можно на странице aws.amazon.com/ec2/instance-types. Например, на момент написания этой книги ни один провайдер, кроме EC2, не предлагает экземпляры Mac, основанные на Mac mini. Для Linux доступны самые разные типы экземпляров — от недорогих до высокопроизводительных, в зависимости от ваших потребностей.
- Для хранения данных Amazon предлагает решение **EBS (Elastic Block Store)** с твердотельными или магнитными накопителями. Их объем и другие характеристики можно тонко настроить в соответствии с вашими задачами. EC2 — более гибкое решение по сравнению со многими альтернативами; у него простой и понятный интерфейс, и вы платите только за то время и ресурсы, которые используете. Пример развертывания в EC2 будет приведен в главе 15 «Как развертывать Linux в облаке с помощью AWS и Azure».

Виртуальные машины Microsoft Azure

Microsoft — второй по величине игрок на рынке облачных вычислений, который предлагает облачную платформу под брендом Azure. Несмотря на то что в качестве облачного провайдера выступает компания Microsoft, самой распространенной операционной системой в Azure является Linux (см. zdnet.com/article/microsoft-developer-reveals-linux-is-now-more-used-on-azure-than-windows-server).

В категории IaaS Azure предоставляет виртуальные машины, которые похожи на экземпляры Amazon EC2 и делятся на разные уровни: от экономичных систем с возможностью расширения до мощных платформ с оптимизированной памятью. Однако Microsoft предлагает другую модель ценообразования: вы либо

платите по часам, либо резервируете экземпляры на один или три года. Почасовая оплата может в итоге выйти дороже, чем резервирование, поэтому осмотрительно выбирайте тарифный план. Интерфейс Microsoft совсем не похож на Amazon и освоить его, на наш взгляд, не самая простая задача.

Из дистрибутивов Linux в Azure можно выбрать различные варианты RHEL, SUSE и Ubuntu.

Azure также предлагает очень мощные решения SaaS, что может стать важным преимуществом, если вы используете другие службы Azure. Пример развертывания в Azure будет приведен в главе 15 «Как развертывать Linux в облаке с помощью AWS и Azure».

Альтернативные предложения IaaS

DigitalOcean — еще один важный игрок на облачном рынке с сильными решениями IaaS. Его виртуальные машины называются **дроппетами**. У DigitalOcean понятный интерфейс, с помощью которого можно быстро настроить облако. Эта процедура состоит из таких шагов:

- Выберите образ (дистрибутив Linux).
- Выберите тарифный план в зависимости от своих требований к виртуальным процессорам, памяти и дисковому пространству.
- Добавьте блоки хранилища.
- Выберите регион центра обработки данных, метод аутентификации (пароль или ключ SSH) и имя узла.
- Можно также назначить дроппеты определенным проектам, которыми вы управляете.

Интерфейс DigitalOcean удобнее, чем у многих конкурентов. Существуют и другие провайдеры IaaS — например, **Linode** и **Hetzner**, которые тоже позволяют управлять виртуальными серверами с помощью легковесного и дружелюбного интерфейса.

Виртуальные машины Linode так и называются — **линодами**, а интерфейс провайдера по простоте использования и дизайну находится где-то между DigitalOcean и Azure.

Hetzner — облачный провайдер со штаб-квартирой в Германии, который занимает существенную долю европейского рынка. Эта компания предлагает выверенный баланс между ресурсами и стоимостью, а спектр ее услуг в целом не уступает крупнейшим конкурентам. Интерфейс Hetzner похож на DigitalOcean: его легко изучить, а облачный экземпляр будет развернут в считанные секунды.

В сегменте, где работают DigitalOcean, Linode и Hetzner, в конце 2010-х появилось предложение от Amazon, которое называется **Lightsail** и предлагает клиентам простой и относительно недорогой способ развертывать виртуальные

частные сервера или виртуальные машины в облаке. По части интерфейса эта служба мало отличается от конкурентов, но при этом обеспечивает надежность инфраструктуры Amazon. Lightsail предлагает несколько дистрибутивов Linux, а также пакеты приложений и инструменты для простого развертывания. Это решение подходит для начинающих пользователей, которые хотят быстро и безопасно развертывать свои веб-приложения.

Существуют и другие продукты IaaS: например, соответствующая платформа от Google называется **Google Cloud Platform (GCP)**, а ее интерфейс очень похож на Azure.

Одна из интересных особенностей GCP заключается в том, что когда вы удаляете проект, эта операция выполняется не сразу, а по расписанию через месяц. Это можно рассматривать как подстраховку на случай, если вы удалили проект по ошибке и хотите вернуть его назад.

В следующем разделе мы подробно рассмотрим некоторые решения PaaS.

PaaS: платформа как услуга

PaaS — это еще одна разновидность облачных вычислений. В отличие от IaaS, она предоставляет аппаратный уровень вместе с прикладным. Оборудование и программное обеспечение размещается у облачного провайдера, и клиенту не нужно ими управлять. Службами PaaS в основном пользуются разработчики приложений, а в качестве провайдеров выступают те же компании, которые предлагают решения IaaS. Основные поставщики PaaS — Amazon, Microsoft и Google. В следующих разделах мы перечислим соответствующие продукты.

Amazon Elastic Beanstalk

Amazon предлагает службу **Elastic Beanstalk** с очень простым интерфейсом, с помощью которого вы можете создать приложение по образцу или загрузить свое собственное, а Beanstalk позаботится обо всем остальном — от деталей развертывания до балансировки нагрузки, масштабирования и мониторинга. Вы выберете, на каких аппаратных экземплярах AWS EC2 будет развернуто приложение.

Google App Engine

Решение PaaS от Google называется Google App Engine: это полностью управляемая бессерверная среда, которая относительно проста в использовании и поддерживает все основные языки веб-программирования, в том числе Go, Node.js, Python, .NET и Java. Это масштабируемое решение с автоматическими обновлениями безопасности, управляемой инфраструктурой и мониторингом, а также подключением к облачным хранилищам Google Cloud. Google предла-

гает конкурентоспособные цены и примерно такой же интерфейс, как для своих продуктов IaaS.

App Platform от DigitalOcean

DigitalOcean предлагает решение PaaS с простым интерфейсом и привлекательной ценовой политикой, которое называется **App Platform**. В нем предусмотрено прямое соединение с вашим репозиторием GitHub или GitLab и обеспечивается полностью управляемая инфраструктура. DigitalOcean находится на одном уровне с такими крупными игроками, как Amazon и Google, и App Platform позволяет управлять инфраструктурой, оснащением, базами данных, средами выполнения и зависимостями приложений, а также нижележащей операционной системой. Продукт поддерживает популярные языки программирования и фреймворки, в том числе Python, Node.js, Django, Go, React и Ruby. App Platform использует открытые облачные стандарты, обеспечивает автоматический анализ кода, позволяет создавать контейнеры и оркестровывать их. Отличительная особенность этого решения — бесплатный *начальный тарифный план*, на котором можно развернуть до трех статических веб-сайтов. *Основной тарифный план* позволяет прототипировать динамические веб-приложения, а *профессиональный план* — разворачивать и выводить на рынок профессиональные приложения. DigitalOcean отличается приятным интерфейсом, с помощью которого новые пользователи могут быстро приступить к работе.

Решения PaaS с открытым исходным кодом

Помимо готовых решений от провайдеров, о которых шла речь раньше, существуют службы PaaS с открытым исходным кодом от **Cloud Foundry**, **Red Hat OpenShift**, **Heroku** и других поставщиков. Три перечисленных продукта заслуживают того, чтобы рассказать о них подробнее:

- **Red Hat OpenShift** — контейнерная платформа для развертывания приложений. Она основана на RHEL и дополнена контейнерной средой выполнения и механизмами для работы с сетью, реестром, аутентификацией и мониторингом. OpenShift разрабатывался как практичное гибридное решение PaaS с полной интеграцией Kubernetes. (Мы кратко представим Kubernetes в следующем разделе «SaaS: контейнеризация как услуга», а подробнее поговорим о нем в главе 16 «Как развертывать приложения с помощью Kubernetes».) OpenShift приобрел CoreOS (о которой речь пойдет ниже), благодаря чему может предложить несколько уникальных решений. В частности, он интегрировал контейнерную платформу CoreOS Tectonic, чтобы предложить пользователям лучшее из обоих продуктов.
- **Cloud Foundry** — облачная платформа с открытым исходным кодом, которая позиционируется как решение PaaS корпоративного класса. Ее можно развернуть на различных инфраструктурах — от собственного корпоративного сервера до таких провайдеров IaaS, как GCP, AWS, Azure или OpenStack. Доступ-

ны различные фреймворки для разработчиков и выбор сертифицированных платформ Cloud Foundry — например, Atos Cloud Foundry, IBM Cloud Foundry, SAP Cloud Platform, SUSE Cloud Application Platform и VMware Tanzu.

- **Heroku** — проект компании Salesforce, который создавался как инновационное решение PaaS. Он опирается на контейнерную систему Dynos с контейнерами под управлением Linux и специальной системой управления контейнерами, которая отличается масштабируемостью и гибкостью. Heroku предлагает полностью управляемые службы данных, которые поддерживают PostgreSQL, Redis, Apache Kafka, а также Heroku Runtime — компонент, который отвечает за оркестровку, масштабирование и управление конфигурацией контейнеров. Heroku также поддерживает множество языков программирования, в том числе Node.js, Ruby, Python, Go, Scala, Clojure и Java.

В сегменте PaaS существует множество решений, которые помогают разработчикам создавать и развертывать приложения и снимают с них бремя управления инфраструктурой. Как вы наверняка убедились, большинство продуктов, о которых шла речь в этом разделе, основаны на контейнерных технологиях. Поэтому в следующем разделе мы подробно рассмотрим CaaS — подмножество IaaS, которое ориентируется на оркестровку контейнеров и специализированные операционные системы для них.

CaaS: контейнеризация как услуга

CaaS — это подмножество модели облачных услуг IaaS. Эта служба позволяет клиентам использовать отдельные контейнеры, кластеры и приложения поверх инфраструктуры, которой управляет провайдер. В зависимости от потребностей заказчика CaaS можно использовать в локальной или облачной среде. В этой модели облачный провайдер предоставляет контейнерные движки и средства оркестровки и управляет ими, а пользователи взаимодействуют с контейнерами либо через API, либо через веб-интерфейс. Платформа оркестровки, которую использует провайдер (чаще всего это Kubernetes в сочетании с Docker), играет важную роль и является ключевым отличием между различными решениями.

Мы рассказывали о контейнерах и виртуальных машинах в главе 11 «Как работать с виртуальными машинами» и в главе 12 «Как управлять контейнерами с помощью Docker», но не вдавались в детали оркестровки или микрооперационных систем, которые специфичны для контейнеров. Сейчас мы расскажем об этом подробнее.

Введение в Kubernetes

Kubernetes — это проект компании Google с открытым исходным кодом, который предоставляет возможность автоматически развертывать и масштабировать контейнерные приложения. Он был написан на языке **Go**. Название «Kubernetes» происходит из греческого языка и обозначает рулевого на корабле. Kubernetes по-

зволяет автоматизировать управление контейнерами, абстрагирует инфраструктуру и обеспечивает мониторинг служб.

Многие новички путают Kubernetes с Docker. На самом деле эти инструменты дополняют друг друга, и каждый из них служит для своих целей. Docker создает контейнер, в котором вы развертываете свое приложение, а Kubernetes координирует контейнеры, когда приложения уже упакованы в них и развернуты. Kubernetes предоставляет функции, которые необходимы для работы контейнеров, — например, он позволяет обнаруживать службы и балансировать нагрузку, оркестрировать хранилища, обеспечивать автоматическое резервное копирование и самовосстановление, а также конфиденциальность. Архитектура Kubernetes состоит из нескольких компонентов, о которых необходимо знать любому администратору. Мы рассмотрим их в следующем разделе.

Компоненты Kubernetes

Как правило, Kubernetes запускают для того, чтобы управлять кластерами узлов, которые обычно представляют собой контейнеры под управлением Linux. Kubernetes состоит из таких основных компонентов:

- **Кластер** — это основная единица обслуживания Kubernetes, потому что задача этого продукта состоит именно в том, чтобы управлять множеством кластеров. Кластер состоит как минимум из управляющего уровня и одного или нескольких узлов, на каждом из которых запускаются контейнеры в модулях («пóдах»).
- **Управляющий уровень** состоит из процессов, которые управляют узлами:

kube-apiserver	Сервер API на фронтенде управляющего уровня
etcd	Хранилище данных внутри кластера в формате «ключ — значение»
kube-scheduler	Ищет модули, которым не назначен узел, и подключает их к узлу, чтобы запустить
kube-controller-manager	Запускает процессы контроллеров, включая контроллеры узлов, репликации, окончечных точек и токенов
cloud-controller-manager	Связывает ваш кластер с API облачного провайдера; включает в себя контроллеры узлов, маршрутизации и служб

- **Узел** — это виртуальная или физическая машина, на которой запущены службы, необходимые для работы модулей. Компоненты узлов работают на каждом узле и поддерживают запущенные модули. Эти компоненты таковы:

kube-proxy	Отвечает за сетевые правила на каждом узле
kubelet	Обеспечивает, чтобы каждый контейнер был запущен внутри модуля

- **Модули** — это набор различных контейнеров, которые работают в кластере. Они являются компонентами рабочей нагрузки.

Освоить кластеры Kubernetes чрезвычайно сложно. Чтобы разобраться в соответствующих понятиях и принципах, нужно много практики и самоотдачи. Кроме того, Kubernetes не делает все за вас; вам самостоятельно придется выбирать контейнерную среду выполнения (поддерживаются **Docker**, **containerd** и **CRI-O**), инструменты CI/CD, средства хранения данных, механизм управления доступом и службы приложений.

Управление кластерами Kubernetes выходит за рамки этой главы, но подробнее рассматривается в главе 16 «Как развертывать приложения с помощью Kubernetes». Надеемся, что из этого краткого введения вы получили представление о понятиях и инструментах, которые использует Kubernetes.

Помимо Kubernetes, существуют и другие решения для оркестровки контейнеров, такие как **Docker Swarm**, **Apache Mesos** и **Nomad** от HashiCorp. Это чрезвычайно мощные инструменты, у которых много пользователей во всем мире. В этой книге мы не будем подробно их рассматривать, а ограничимся только этим кратким перечислением. В следующем разделе мы в общих чертах поговорим о контейнерных решениях в облаке.

Развертывание контейнеров в облачной среде

Чтобы оркестровывать контейнеры в облачной среде, вам пригодятся такие продукты:

- **Amazon Elastic Container Service (ECS)** — это полностью управляемая служба для оркестровки контейнеров. Она предлагает дополнительное бессерверное решение **AWS Fargate** и работает в контексте ключевых служб Amazon, а это значит, что решение протестировано и безопасно для массового использования.
- **Amazon Elastic Kubernetes Service (EKS)** позволяет оркестрировать приложения Kubernetes. Эта служба основана на **Amazon EKS Distro (EKS-D)** — это дистрибутив Kubernetes, который разработала компания Amazon на основе оригинального открытого кода Kubernetes. С помощью EKS-D можно запускать Kubernetes как на локальном оборудовании, так и на собственных экземплярах Amazon EC2 или на виртуальных машинах VMware vSphere.
- **Google Kubernetes Engine (GKE)** предлагает готовые шаблоны развертывания, которые позволяют автоматически масштабировать модули в зависимости от потребления процессора и памяти. Доступно масштабирование в рамках нескольких пулов, а GKE Sandbox обеспечивает дополнительный уровень безопасности, защищая ядро узла и запущенные приложения.
- **Microsoft Azure Kubernetes Service (AKS)** — это управляемая служба, которая позволяет развертывать кластеры контейнерных приложений. Как и другие провайдеры, Microsoft предлагает полностью управляемое решение, обеспечи-

вая обслуживание ресурсов и мониторинг состояния. Узлы AKS запускаются на виртуальных машинах Azure и поддерживают различные операционные системы, например образы Microsoft Windows Server. Кроме того, доступны бесплатные обновления до наиболее свежих образов Kubernetes. Среди прочих решений AKS предлагает узлы с поддержкой графических процессоров, тома хранилищ и интеграцию средств разработки с Visual Studio Code.

После того как мы познакомились с некоторыми решениями для развертывания Kubernetes в облаке и узнали об основных компонентах Kubernetes, в следующем разделе мы поговорим о том, какую роль играют микросервисы в облачных вычислениях.

Микросервисы в облачных вычислениях

Микросервис — это архитектурный стиль, который используется в доставке приложений. Со временем благодаря развитию облачных технологий их доставка эволюционировала от монолитной к децентрализованной модели. После исторического старта AWS в 2006 году, за которым последовал запуск Heroku в 2007-м и Vagrant в 2010-м, развертывание приложений тоже начало меняться, чтобы использовать преимущества новых облачных решений. Приложения перешли от единой, большой и монолитной кодовой базы к модели, в которой каждое приложение обращается к отдельным наборам служб. В результате кодовая база становится легче и опирается на многочисленные внешние службы.

Давайте разберемся, чем **монолитное приложение** отличается от приложения с **микросервисной архитектурой**:

- Монолитная модель содержит все функциональные возможности в рамках одного процесса и развертывается путем простой репликации на нескольких серверах. А микросервисная архитектура предполагает, что функциональные возможности приложения разделены на различные службы, которые распределяются и масштабируются по разным серверам в зависимости от потребностей пользователей.
- Микросервисная архитектура основана на модульном подходе, где каждый модуль соответствует определенной службе. Службы функционируют независимо друг от друга и взаимодействуют через REST API поверх протокола HTTP. Это значит, что разные функции приложения могут реализовываться на разных языках в зависимости от того, какой лучше подходит. Эта модульная схема также эффективно сочетается с новыми контейнерными технологиями.
- Микросервисная архитектура позволяет быстро разрабатывать сложные приложения. Она не привязана к конкретным технологиям или языкам, и благодаря ей можно независимо масштабировать и обновлять каждую службу и компонент, не нарушая работу других служб. Микросервисная архитектура считается весьма отказоустойчивой. Чтобы перевести существу-

ющие монолитные приложения на микросервисную архитектуру, их можно разбить на отдельные модульные службы. При этом не нужно переписывать все приложение, а достаточно лишь разбить его кодовую базу на более мелкие части. Благодаря модульному подходу микросервисы оптимизированы для DevOps и CI/CD.

В следующем разделе вы познакомитесь с практиками и инструментами DevOps.

Что такое DevOps

DevOps — это целая культура. В ее названии сочетаются слова *development* (разработка) и *operations* (эксплуатация), и она объединяет практики и инструменты, которые помогают обеспечивать быструю доставку. DevOps — это скорость и гибкость. Известная фраза «Время — деньги» актуальна и в секторе IT: если вы быстро доставляете клиентам свои услуги и приложения, это может стать ключевым фактором успеха вашего бизнеса, а медлительность грозит его погубить.

DevOps — это модель сотрудничества между различными командами, которые участвуют в доставке служб и приложений. Это значит, что все команды в равной степени задействованы на каждом этапе жизненного цикла продукта — от разработки и тестирования до развертывания и управления. DevOps предполагает, что ни одна команда не работает в замкнутой среде, а все действуют прозрачно, чтобы достичь маневренности, которая необходима для успеха. Существует разновидность DevOps, которая называется **DevSecOps** и в которой команды безопасности и обеспечения качества в равной степени участвуют в цикле разработки.

Важнейший элемент модели DevOps — автоматизированные процессы, которые налаживаются с помощью специальных инструментов. Соответствующий образ мышления, ориентированный на гибкость и скорость, породил новое понятие — CI/CD (непрерывная интеграция и непрерывное развертывание). Идеология CI/CD подразумевает, что каждый этап разработки происходит непрерывно, без остановок. Появились новые инструменты автоматизации, которые поддерживают эту идеологию; пожалуй, самый известный из них — программная система **Jenkins** с открытым исходным кодом, которая написана на Java и позволяет автоматизировать процессы разработки программного обеспечения, начиная со сборки и тестирования и заканчивая доставкой. Это модульное решение, которое можно расширять с помощью плагинов. Вокруг Jenkins сформировалась обширная экосистема, и на выбор предлагаются сотни плагинов. Одно из преимуществ Jenkins состоит в том, что с помощью специальных плагинов он позволяет настраивать конвейеры. Конвейер — это инструмент, который внедряет в жизненный цикл приложения непрерывную интеграцию как автоматизированный процесс. Jenkins можно использовать как на локальном оборудовании, так и в облаке. Кроме того, это решение эффективно выступает в качестве службы SaaS.

Философия DevOps касается не только развертывания приложений. Отлаженные процессы CI/CD тесно связаны с состоянием инфраструктуры. Управлять инфраструктурой, которая поддерживает развертывание приложений, помогают такие системы управления облачными средами, как Ansible, Puppet и Chef. Именно поэтому конфигурация и управление на уровне инфраструктуры чрезвычайно важны. В следующем разделе пойдет речь об управлении облачной инфраструктурой.

Инструменты управления облачными средами

В наши дни разработка и развертывание программного обеспечения опираются на множество физических систем и виртуальных машин. Управлять всеми соответствующими средами для разработки, тестирования и выпуска в эксплуатацию — непростая задача, которая не обходится без автоматизированных инструментов. К самым распространенным решениям для управления облачной инфраструктурой относятся такие системы, как **Ansible**, **Puppet** и **Chef Infra**. Все это мощные и надежные средства, и в этом разделе мы кратко представим каждое из них, а с Ansible глубже познакомимся на практике в главе 17 «Как управлять инфраструктурой и автоматизацией с помощью Ansible».

Ansible

Ansible — это проект с открытым исходным кодом на Python, который сейчас поддерживает компания Red Hat. Он считается простым инструментом автоматизации, с помощью которого можно выполнять различные операции: например, развертывать приложения, управлять конфигурацией, оснащать облачные среды и оркестрировать службы. Чтобы определять категории систем, Ansible использует абстракцию узлов: **управляющий узел** — это главная машина, на которой работает Ansible, а различные **управляемые узлы** — это остальные машины, которые контролирует главный узел. Все узлы подключаются по SSH и управляются через приложения, которые называются **модулями Ansible**. Каждый модуль выполняет на управляемых узлах определенную задачу, и когда она завершена, модуль удаляется с соответствующего узла.

Ansible считается одним из самых простых инструментов автоматизации. Порядок использования модулей определяется в сценарии («плейбуке») Ansible. Сценарии пишутся на языке **YAML**, который в основном применяется для конфигурационных файлов. Ansible также использует **описи**, в которых хранятся списки управляемых узлов. Когда вы запускаете команды на узлах, их можно применять на основе списков из описи в соответствии с **шаблонами**. Ansible применяет команды к каждому узлу или группе узлов, которые доступны в том или ином шаблоне. На клиентских машинах поддерживается Linux/Unix и Windows, но на управляющем узле должна быть Linux/Unix.

Puppet

Puppet — один из старейших инструментов автоматизации, и его архитектура отличается от Ansible прежде всего тем, что Puppet выглядит гораздо сложнее. В этой системе используются понятия **первичных серверов** и **агентов**. Puppet работает с инфраструктурным кодом, который пишется на специализированном предметно-ориентированном языке на основе Ruby. Код возникает на первичном сервере, передается агенту, а затем преобразуется в команды, которые выполняются на управляемой системе. В Puppet также есть инструмент инвентаризации, который называется **Factor** и хранит сведения об агентах, такие как имя узла, IP-адрес и операционная система. Сохраненная информация отправляется обратно на первичный сервер в виде **манифеста**, который затем преобразуется в документ JSON — так называемый **каталог**. Все манифесты хранятся внутри **модулей**; это инструменты, каждый из которых предназначен для определенной задачи. Каждый модуль содержит информацию в виде кода и данных. Эти данные централизованы и управляются системой под названием **Hiera**. Все данные, которые производит Puppet, хранятся в базах данных, и любое приложение, которое намерено ими управлять, может делать это с помощью API. Первичный сервер Puppet должен работать под управлением Linux/Unix.

Chef Infra

Chef Infra — еще один инструмент автоматизации с клиент-серверной архитектурой. Его ключевые элементы называются **поваренными книгами** и **рецептами**. Конфигурация Chef состоит из таких компонентов:

- **Сервер Chef** — это своего рода концентратор, который обрабатывает все конфигурационные данные и в основном используется для того, чтобы загружать поваренные книги в Chef Client.
- **Chef Client** — это приложение, которое устанавливается на каждый узел управляемой инфраструктуры.
- **Рабочая станция Chef** управляет поваренными книгами, которые позволяют администрировать инфраструктуру.

В Chef Infra используется предметно-ориентированный язык на основе Ruby, подобный языку Puppet. Сервер должен работать под управлением Linux/Unix, а клиенты поддерживают также и Windows.

Все инструменты автоматизации и конфигурации, о которых шла речь в этом разделе, устроены по-разному с точки зрения архитектуры, но делают одно и то же — предоставляют уровень абстракции, который описывает желаемое состояние инфраструктуры. У каждого инструмента есть свои особенности, а также свои сильные и слабые стороны. Chef Infra и Puppet сложнее изучить — виной тому во многом их собственные предметно-ориентированные языки, — в то время как Ansible легче освоить благодаря тому, что у него относительно простая

архитектура, а сценарии пишутся на YAML. Тем не менее все эти продукты одинаково эффективно выполняют свои функции.

Итоги

В этой главе вы познакомились с облачными вычислениями и получили представление о наиболее важных понятиях, инструментах и решениях в этой сфере. Этого фундамента должно хватить для того, чтобы вы продолжили самостоятельно изучать такую обширную и сложную тему, как облачные технологии. Чтобы получить более подробную информацию, обратитесь к разделу «Дополнительная литература».

Мы поговорили о стандартах облачных вычислений — важной теме, которую часто упускают из вида, — а также об основных типах облачных решений и служб. Теперь вы в курсе, что представляют собой разные решения «X как услуга» и чем они отличаются друг от друга. Вы познакомились с ведущими продуктами этого сегмента и узнали, как их предоставляют основные игроки в этой области: Amazon, Google и Microsoft. Кроме того, мы обзорно поговорили о том, как оркестрируются контейнеры с помощью Kubernetes. Вы узнали об API и минималистичных операционных системах, которые специфичны для контейнеров, а также о культуре DevOps, микросервисах и средствах автоматизации инфраструктуры. В этой главе было много нового материала, но не забывайте, что наш обзор затронул лишь самые поверхностные вопросы облачных вычислений.

В следующей главе мы поговорим об облачном развертывании с более практической точки зрения. Вы узнаете, как развернуть Linux в основных облачных средах — таких, как AWS, Azure и GCP.

Дополнительная литература

Если вы хотите узнать больше об облачных технологиях, обратите внимание на следующие издания:

- Бен Сильверман (Ben Silverman), Майкл Солберг (Michael Solberg). «OpenStack for Architects», Second Edition.
- Микаэл Криф (Mikael Krief). «Learning DevOps», Second Edition.
- Мехмет Озкайя (Mehmet Ozkaya). «Design Microservices Architecture with Patterns and Principles», видеокурс.
- Йерун Мюлдер (Jeroen Mulder). «Multi-Cloud Strategy for Cloud Architects», Second Edition.
- Сафир СМ (Safeer CM). «Architecting Cloud-Native Serverless Solutions».

15

Как развертывать Linux в облаке с помощью AWS и Azure

В последние годы многие службы и продукты переходят с локальных вычислительных платформ на частные и публичные облачные среды. В мире, который постоянно меняется и ускоряется, компаниям жизненно необходимо развертывать и запускать приложения в высокомасштабируемой, эффективной и безопасной инфраструктуре. Чтобы оставаться на локальных вычислительных ресурсах и при этом поддерживать необходимый уровень безопасности и производительности, необходимы затраты (в том числе на услуги квалифицированных специалистов), которые едва ли оправданны по сравнению с современными предложениями публичных облаков. Малые и крупные компании и рабочие группы все чаще переходят на облачные решения, хотя крупные корпорации делают это сравнительно медленно.

Облачные вычисления удобно метафорически представлять себе как прикладные службы, которые отпускаются «*на разлив*». Вам нужно больше ресурсов для ваших приложений? Просто *откройте кран* и создайте столько виртуальных машин или экземпляров, сколько нужно (горизонтальное масштабирование), либо добавьте процессоры или память (вертикальное масштабирование). Когда вы набрали достаточно ресурсов, просто *закройте кран*.

Публичные облачные провайдеры предоставляют все эти функции по относительно низким ценам, благодаря чему не надо тратить на то, чтобы обслуживать локальную инфраструктуру с аналогичными функциями.

В этой главе вы познакомитесь с **Amazon Web Services (AWS)** и **Microsoft Azure** — двумя основными провайдерами публичных облачных услуг, а также получите практические рекомендации по поводу того, как развертывать приложения в облаке. В частности, мы сосредоточимся на типичных процедурах по управлению облаком с помощью административной веб-консоли и интерфейса командной строки.

К концу этой главы вы научитесь использовать веб-консоль и инструменты командной строки AWS и Azure, чтобы управлять облачными ресурсами двух самых популярных провайдеров. Вы также сможете принимать взвешенные

решения о том, как найти разумный баланс между производительностью и стоимостью, когда настраиваете и запускаете свои ресурсы в облаке.

Мы надеемся, что для администраторов Linux — как начинающих, так и опытных — содержание этой главы окажется актуальным и познавательным. Мы рассмотрим, как работать с AWS и Azure с чисто практической точки зрения, но воздержимся от того, чтобы сравнивать плюсы и минусы этих систем.

В AWS и Azure много общих (или очень похожих) понятий и принципов, но мы не будем повторять соответствующий материал дважды. Давайте вспомним, что AWS первым проложил путь в сферу публичных облачных служб, а другие провайдеры последовали за ним, перенимая и порой улучшая основные парадигмы и рабочие процессы. Поэтому мы начнем с AWS и в контексте этой службы подробнее рассмотрим общие принципы оснащения облачных экземпляров (например, регионы и **зоны доступности**), которые в похожей форме реализованы в Azure.

Окончательный выбор между AWS и Azure оставляем на ваше усмотрение. В этой главе мы даем вам карту, а дорогу выбираете вы сами.

Технические требования

Чтобы выполнять упражнения из этой главы, вам понадобятся такие компоненты:

- Учетные записи AWS и Azure. Оба облачных провайдера предоставляют бесплатные подписки:

AWS	aws.amazon.com/free
Microsoft Azure	azure.microsoft.com/free

- Локальный компьютер с любым дистрибутивом Linux, на котором вы будете экспериментировать с утилитами командной строки AWS и Azure.
- Современный браузер (например, Google Chrome или Mozilla Firefox) для того, чтобы управлять AWS и Azure с помощью веб-консоли. Доступ к этой консоли можно получить с любого устройства, на котором установлен браузер.
- Терминал командной строки Linux и основные навыки работы с командной оболочкой для того, чтобы запускать команды AWS и Azure.

Не забывайте выключать свои экземпляры EC2 или виртуальные машины Azure после того, как вы закончили упражнения, иначе вам могут выставить довольно высокие счета.

Итак, давайте начнем с первого облачного решения — AWS EC2.

AWS EC2

AWS Elastic Compute Cloud (EC2) — это масштабируемая вычислительная инфраструктура, где пользователи могут арендовать виртуальные вычислительные платформы и службы для того, чтобы запускать свои облачные приложения. В последние годы AWS EC2 завоевала огромную популярность благодаря выдающейся производительности и масштабируемости в сочетании с относительно экономичными тарифными планами. В этом разделе вы получите элементарные практические знания о том, как начать развертывать экземпляры AWS EC2 и управлять ими, чтобы запускать свои приложения. В частности, вы познакомитесь с различными типами экземпляров и узнаете, как различать уровни оснащения и соответствующие тарифные планы, как подключаться к экземплярам с помощью SSH и передавать файлы с помощью SCP, а также как работать с командами AWS.

К концу этого раздела вы получите базовое представление об AWS EC2 и сможете выбрать и развертывать экземпляры EC2, а также управлять ими.

Как создавать и оснащать экземпляры AWS EC2

AWS EC2 предлагает несколько типов экземпляров, которые различаются оснащением, мощностью, стоимостью и сценариями использования, и между ними не всегда просто сделать выбор. В этом разделе мы кратко опишем плюсы и минусы разных типов, а также расскажем, как выбрать оптимальное решение с точки зрения стоимости. Кроме того, мы продемонстрируем, как запустить экземпляр каждого типа с помощью консоли AWS.

Типы экземпляров EC2 можно оценивать по двум параметрам: оснащение и стоимость. Когда вы будете принимать решение о том, какой тип выбрать, придется учитывать оба этих фактора. Давайте их рассмотрим.

- **Оснащение** — вычислительная мощность экземпляра EC2, которая выражается в количестве и характеристиках процессоров (реальных или виртуальных), а также в объеме памяти и хранилища.

Некоторые типы экземпляров EC2 также предоставляют вычислительные возможности **графических процессоров** или **программируемых вентильных матриц (FPGA)**. Здесь мы не будем подробно рассматривать все варианты оснащения, но вы можете ознакомиться с ними на официальных страницах:

- docs.aws.amazon.com/AWSEC2/latest/UserGuide/instance-types.html
- aws.amazon.com/ec2/instance-types

- **Стоимость.** На момент написания этой книги EC2 предлагает четыре ценовые модели:
 - **Экземпляры по требованию (on-demand).** Вы платите за потребляемую вычислительную мощность в секунду, без долгосрочных обязательств.

- **Зарезервированные экземпляры** (reserved) позволяют значительно сэкономить по сравнению с экземплярами по требованию, если вы арендуете ресурсы на длительный срок.
- **Точечные экземпляры** (spot) работают на высвобождающихся мощностях и стоят дешевле, чем экземпляры по требованию.
- **Выделенные экземпляры** (dedicated) функционируют в **виртуальном частном облаке (VPC)**, которое закреплено за учетной записью одного плательщика.

В следующих разделах мы рассмотрим все эти типы экземпляров и покажем, как их запустить.

Однако прежде чем создавать экземпляр, давайте познакомимся с еще одним ключевым понятием EC2 — **зонами доступности**.

Зоны доступности EC2

Сервис AWS EC2 доступен в нескольких десятках **регионов** по всему миру, например:

США — Запад	Орегон (США)	us-west-2
Азиатско-Тихоокеанский регион — Юг	Мумбаи (Индия)	ap-south-1

В каждом регионе определено несколько зон доступности, которые по сути являются центрами обработки данных. Базовые инфраструктуры разных регионов полностью изолированы друг от друга, чтобы обеспечить отказоустойчивость и высокую доступность. Если регион станет недоступным, это затронет только его экземпляры EC2, а экземпляры в других регионах продолжат работать без перебоев.

Аналогичным образом устроены зоны доступности, которые предоставляют высокодоступные и отказоустойчивые службы EC2 в пределах одного региона. Когда вы запускаете экземпляр EC2, он создается в текущем регионе, который вы выбрали в консоли AWS. Когда администратор EC2 управляет экземплярами, он может переключаться между различными регионами, и в административной консоли будут отображаться только экземпляры в выбранном регионе. Администратор обычно выбирает регион в зависимости от того, где географически находятся пользователи, которые обращаются к экземпляру EC2.

Теперь, когда у вас есть общее представление о различных типах экземпляров EC2, давайте рассмотрим экземпляры по требованию.

Экземпляры EC2 по требованию

Экземпляры по требованию AWS EC2 списывают средства *по мере потребления* ресурсов в секунду, без долгосрочных обязательств. Такие экземпляры лучше

всего подходят для того, чтобы экспериментировать с рабочими нагрузками, когда заранее неизвестно, сколько ресурсов потребуется (скажем, в процессе разработки). Однако эта гибкость обходится дороже, чем, например, зарезервированные экземпляры.

Давайте запустим экземпляр по требованию:

1. Войдите в консоль учетной записи AWS (console.aws.amazon.com). На следующем снимке экрана показано, как эта консоль выглядит по умолчанию. На ней представлена информация о пользователе (1), регионе (2) и доступных или недавно посещенных службах (3):

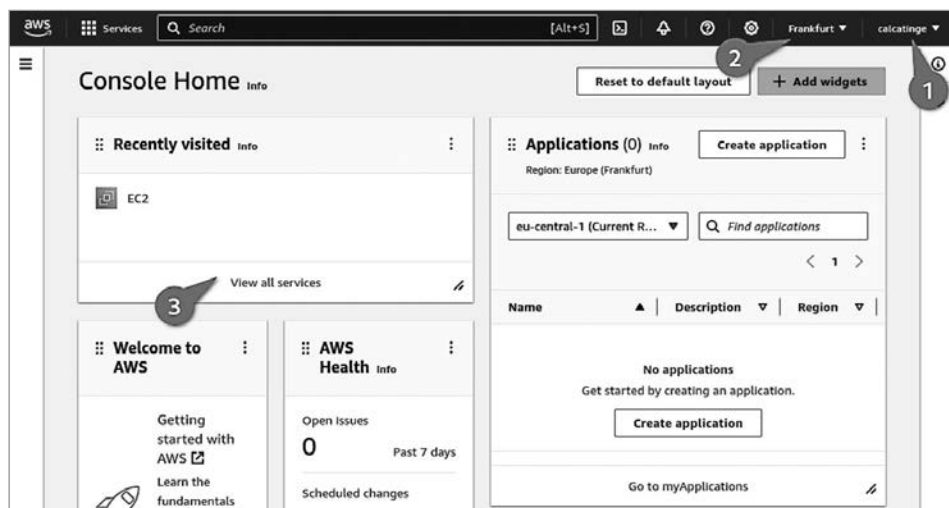


Рис. 15.1. Управляющая консоль AWS

2. Выберите службу EC2 из списка в левой части интерфейса. Вы перейдете в панель управления EC2 (рис. 15.2).

Если нажать на кнопку **Launch Instance** (Запустить экземпляр), начнется пошаговый процесс создания экземпляра по требованию. Вы перейдете на следующий экран, где можно указать основные параметры конфигурации нового экземпляра — в частности, имя, операционную систему, тип экземпляра, ключи для входа, настройки сети и хранилища.

3. Далее присвойте экземпляру имя и тег. В нашем упражнении мы зададим имя `aws_packt_testing_1` (рис. 15.3).

Кроме того, давайте добавим новый тег — для этого нужно щелкнуть на **Add additional tags** (Добавить дополнительные теги) справа от имени экземпляра. Откроется интерфейс, в котором можно создать теги в формате «ключ — значение»: это свойства, которые помечают или идентифицируют экземпляр EC2. Если мы управляем большим количеством экземпляров, теги помогут

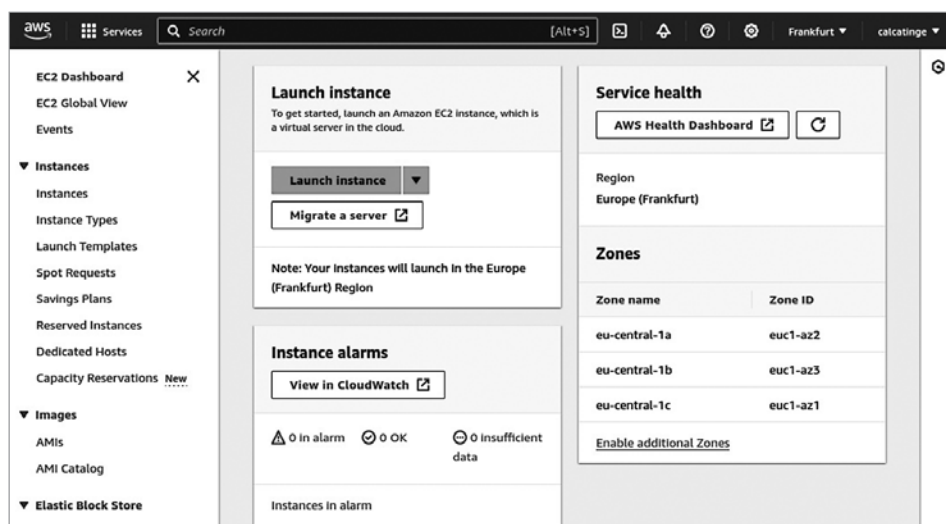


Рис. 15.2. Панель управления EC2

Launch an instance Info

Amazon EC2 allows you to create virtual machines, or instances, that run on the AWS Cloud. Quickly get started by following the simple steps below.

Name and tags Info

Name

aws_packt_testing_1

Add additional tags

Рис. 15.3. Присвоение имени новому экземпляру EC2

их искать и фильтровать. Например, если мы решили идентифицировать наш экземпляр EC2 как часть среды *Packt*, можно создать тег с ключом `env` и значением `packt`. Если нужно, можно присвоить экземпляру и другие теги (рис. 15.4).

4. Выберите операционную систему. В нашем случае мы укажем Ubuntu 22.04 LTS (рис. 15.5).

EC2 предлагает и другие операционные системы, включая macOS, Microsoft Windows и различные дистрибутивы Linux — такие, как RHEL, SUSE и Debian, а также Amazon Linux.

5. Выберите тип экземпляра в зависимости от того, какой объем ресурсов вам необходим. В нашем случае мы выберем тип **t2.micro** с одним виртуальным процессором и 1 Гбайт памяти (рис. 15.6).

▼ Name and tags Info

Key Info

Value Info

Resource types Info

Q Name X

Q aws_packt_testin X

Select resource types ▼

Remove

Instances X

Key Info

Value Info

Resource types Info

Q env X

Q packt X

Select resource types ▼

Remove

Instances X

Add new tag

You can add up to 48 more tags.

Рис. 15.4. Присвоение тегов экземпляру EC2

▼ Application and OS Images (Amazon Machine Image) Info

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. Search or Browse for AMIs if you don't see what you are looking for below

Q Search our full catalog including 1000s of application and OS images

Quick Start

Amazon Linux

macOS

Ubuntu

Windows

Red Hat

SUSE Linux

Browse more AMIs

Including AMIs from AWS, Marketplace and the Community

Amazon Machine Image (AMI)

Ubuntu Server 22.04 LTS (HVM), SSD Volume Type

Free tier eligible ▼

ami-0faab6bdbac9486fb (64-bit (x86)) / ami-0fc02b454efabb390 (64-bit (Arm))

Virtualization: hvm ENA enabled: true Root device type: ebs

Description

Canonical, Ubuntu, 22.04 LTS, amd64 jammy Image build on 2023-12-07

Architecture

AMI ID

Verified provider

64-bit (x86) ▼

ami-0faab6bdbac9486fb

Verified provider

Рис. 15.5. Выбор операционной системы для экземпляра EC2

6. В разделе **Network settings** (Параметры сети) мы пока оставим настройки по умолчанию (рис. 15.7).

▼ **Instance type** Info | Get advice

Instance type

t2.micro
 Family: t2 1 vCPU 1 GiB Memory Current generation: true
 On-Demand Windows base pricing: 0.018 USD per Hour
 On-Demand SUSE base pricing: 0.0134 USD per Hour
 On-Demand Linux base pricing: 0.0134 USD per Hour
 On-Demand RHEL base pricing: 0.0734 USD per Hour

Free tier eligible

☒ All generations
[Compare instance types](#)

Additional costs apply for AMIs with pre-installed software

Рис. 15.6. Выбор типа экземпляра t2.micro

▼ **Network settings** Info Edit

Network Info

vpc-989657f3

Subnet Info

No preference (Default subnet in any availability zone)

Auto-assign public IP Info

Enable

Firewall (security groups) Info

A security group is a set of firewall rules that control the traffic for your instance. Add rules to allow specific traffic to reach your instance.

☒ Create security group
 ☐ Select existing security group

We'll create a new security group called 'launch-wizard-5' with the following rules:

☒ Allow SSH traffic from
 Helps you connect to your instance

Anywhere
0.0.0.0/0

☐ Allow HTTPS traffic from the Internet
 To set up an endpoint, for example when creating a web server

☐ Allow HTTP traffic from the Internet
 To set up an endpoint, for example when creating a web server

⚠ Rules with source of 0.0.0.0/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only.

Рис. 15.7. Настройка сети для экземпляра EC2

- Теперь нужно настроить хранилище. По умолчанию экземпляр EC2 предоставляет накопитель SSD объемом 8 Гбайт, но на бесплатном уровне его можно расширить до 30 Гбайт. Мы будем использовать стандартные 8 Гбайт (рис. 15.8).
- Наконец, перед тем как запускать новый экземпляр EC2, просмотрите подробную сводку и решите, нужны ли еще какие-то изменения. Если вы убедились, что новый экземпляр соответствует вашим потребностям, его можно создать, нажав на кнопку **Launch instance** (Запустить экземпляр) (рис. 15.9).

▼ **Configure storage** Info

Advanced

1x GiB Root volume (Not encrypted)

ⓘ Free tier eligible customers can get up to 30 GB of EBS General Purpose (SSD) or Magnetic storage

×

Add new volume

The selected AMI contains more instance store volumes than the instance allows. Only the first 0 instance store volumes from the AMI will be accessible from the instance

ⓘ Click refresh to view backup information

The tags that you assign determine whether the instance will be backed up by any Data Lifecycle Manager policies.

↻

0 x File systems

Edit

Рис. 15.8. Настройка хранилища для экземпляра EC2

▼ **Summary**

Number of Instances Info

Software Image (AMI)
Canonical, Ubuntu, 22.04 LTS, ...read more
ami-0faab6bdbac9486fb

Virtual server type (Instance type)
t2.micro

Firewall (security group)
New security group

Storage (volumes)
1 volume(s) - 8 GiB

ⓘ Free tier: In your first year includes 750 hours of t2.micro (or t3.micro in the Regions in which t2.micro is unavailable) instance usage on free tier AMIs per month, 30 GiB of EBS storage, 2 million I/Os, 1 GB of snapshots, and 100 GB of bandwidth to the Internet.

×

Cancel

Launch instance

Review commands

Рис. 15.9. Сводная информация о новом экземпляре EC2 и кнопка для запуска

На предыдущих этапах можно было задать и другие, более тонкие настройки, но если этого не сделать (как в нашем случае), то EC2 присвоит им значения по умолчанию.

9. Когда вы запустите экземпляр EC2, панель управления предложит создать или выбрать пару ключей, чтобы удаленно подключаться к экземпляру по SSH. Выберите вариант **Create new key pair** (Создать новую пару ключей). Мы назовем эту пару `packt_aws_key`, выберем тип шифрования **RSA** и формат файла **.pem**, а затем нажмем **Create key pair** (Создать пару ключей) в правом нижнем углу (рис. 15.10).

Система автоматически предложит вам загрузить закрытый ключ в формате `.pem` на свой локальный компьютер.

Create key pair [X]

ⓘ We noticed that you didn't select a key pair. If you want to be able to connect to your Instance it is recommended that you create one.

☒ Create new key pair ☐ Proceed without key pair

Key pair name
Key pairs allow you to connect to your instance securely.

The name can include up to 255 ASCII characters. It can't include leading or trailing spaces.

Key pair type

☒ **RSA**
RSA encrypted private and public key pair

☐ **ED25519**
ED25519 encrypted private and public key pair

Private key file format

☒ **.pem**
For use with OpenSSH

☐ **.ppk**
For use with PuTTY

⚠ When prompted, store the private key in a secure and accessible location on your computer. **You will need it later to connect to your instance.** [Learn more](#)

Cancel **Create key pair**

Рис. 15.10. Создание пары ключей SSH

10. После того как вы сохраните файл (в нашем случае он называется `packt-ec2.pem`) в безопасное место на своем компьютере, его можно будет использовать для того, чтобы подключаться к экземпляру EC2 с помощью команды `ssh`:

```
ssh -i /путь/к/ключу/packt-ec2.pem имя_пользователя@IP_экземпляра
```

Позже в этой главе мы подробнее поговорим о том, как подключиться по SSH к экземпляру EC2.

11. После того как вы создадите и запустите экземпляр EC2, на экране появится кнопка **View Instances** (Просмотреть экземпляры), которая приведет вас на панель управления EC2, где отображаются экземпляры в текущем регионе. Список экземпляров можно отфильтровать по различным свойствам, включая теги. Например, если задать фильтр по тегу `env: packt`, вы увидите сведения о том экземпляре EC2, который только что создали (рис. 15.11).

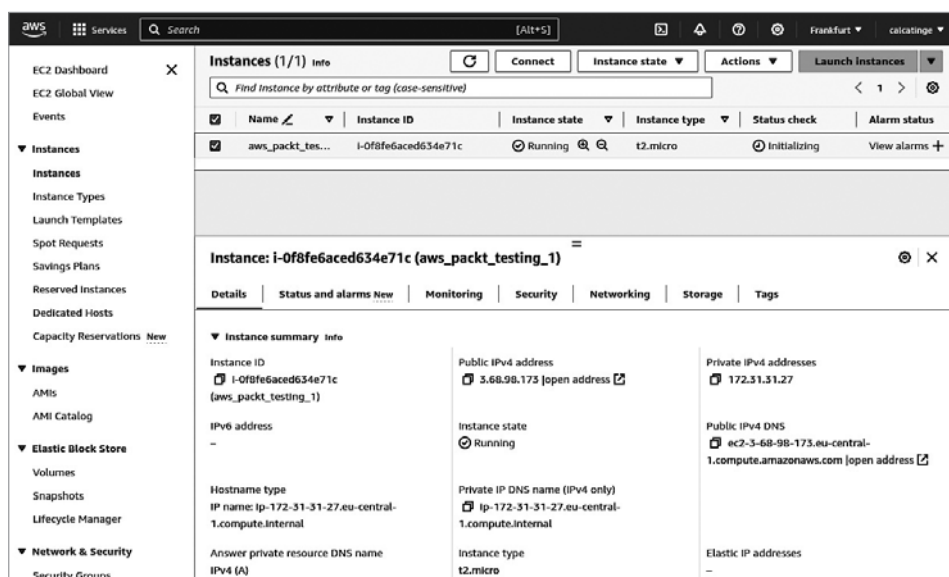


Рис. 15.11. Экземпляр EC2 в режиме эксплуатации

Дополнительную информацию об экземплярах по требованию можно найти на странице docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-on-demand-instances.html.

Теперь, когда вы знаете, как запускать экземпляры по требованию, давайте поговорим о зарезервированных экземплярах.

Зарезервированные экземпляры EC2

Чтобы пользоваться **зарезервированными экземплярами**, нужно арендовать вычислительные мощности EC2 определенного типа на определенный срок,

а именно на 1 или 3 года. На рис. 15.12 показаны основные характеристики, которые нужно задать заранее:

- Platform (Платформа) — например, **Linux/UNIX**.
- Tenancy (Режим аренды). Экземпляр может работать на совместно используемом (по умолчанию) или выделенном оборудовании.
- Offering class (Класс предложения). Экземпляр может быть стандартным или конвертируемым. Основное различие между этими классами состоит в том, что стандартные экземпляры дешевле, но их нельзя заменить на экземпляры с другой конфигурацией.
- Instance Type (Тип экземпляра) — например, **t2.large**.
- Term (Срок) — например, **12 месяцев**.
- Payment option (Режим оплаты): полная предоплата, частичная предоплата или без предоплаты.

Purchase Reserved Instances

☐ Only show offerings that reserve capacity

Platform

Tenancy

Offering class

Linux/UNIX

Default

Any

Instance type

Term

Payment option

t3.micro

1 month to 12 months

Any

Search

Seller	Term	Effective rate	Upfront price	Hourly rate	Payment option	Offering class	Quant available
AWS	12 months	\$0.007	\$62.00	\$0.000	All upfront	Standard	Unlim
AWS	12 months	\$0.009	\$81.00	\$0.000	All upfront	Convertible	Unlim
AWS	12 months	\$0.008	\$0.00	\$0.008	No upfront	Standard	Unlim
AWS	12 months	\$0.010	\$0.00	\$0.010	No upfront	Convertible	Unlim
AWS	12 months	\$0.007	\$32.00	\$0.004	Partial upfront	Standard	Unlim
AWS	12 months	\$0.009	\$42.00	\$0.005	Partial upfront	Convertible	Unlim

You currently have no items in your cart.

Cancel

View cart

Рис. 15.12. Настройка зарезервированного экземпляра EC2

Окончательная стоимость экземпляра будет зависеть от многих параметров, в том числе от того, какой объем облачных ресурсов задействован и в каком режиме вы оплачиваете службу. Например, если вы платите всю сумму вперед,

это получится дешевле, чем при других режимах оплаты. Выбирайте нужный вариант, взвесив факторы экономии и гибкости.

Тарифные планы зарезервированных экземпляров во многом похожи на тарифные планы мобильной связи: вы настраиваете все нужные параметры, а затем устанавливаете определенный срок. По сравнению с экземплярами по требованию зарезервированные экземпляры обходятся значительно дешевле (иногда удастся сэкономить до 75 % средств), но обеспечивают меньше маневренности, если нужно изменить конфигурацию.

Чтобы запустить зарезервированный экземпляр, перейдите на панель управления EC2 в консоли AWS и выберите в разделе **Instances** (Экземпляры) на левой панели пункт **Reserved Instances** (Зарезервированные экземпляры), а затем нажмите **Purchase Reserved Instances** (Приобрести зарезервированные экземпляры).

Дополнительную информацию о зарезервированных экземплярах EC2 можно найти на странице docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-reserved-instances.html.

Итак, мы узнали, что зарезервированные экземпляры EC2 экономически выгоднее, чем экземпляры по требованию. Теперь давайте рассмотрим еще один способ снизить затраты — точечные экземпляры.

Точечные экземпляры EC2

Точечный экземпляр (spot instance) — это неиспользуемый экземпляр, который ожидает аренды. Время, в течение которого он свободен и находится в вашем распоряжении, зависит от того, насколько доступны запрашиваемые мощности в EC2, при условии, что их себестоимость не превышает сумму, которую вы готовы заплатить за точечный экземпляр. AWS рекламирует точечные экземпляры со скидкой до 90 % по сравнению с экземплярами по требованию.

Основной недостаток точечных экземпляров состоит в том, что свободных экземпляров может не оказаться в наличии, если запрошенная мощность больше не доступна по цене, которую вы изначально согласовали. В этом случае экземпляр будет остановлен (и, возможно, его арендует кто-то другой). EC2 любезно предупреждает об этом за 2 минуты до остановки, чтобы вы успели корректно завершить рабочие процессы приложений, которые запущены в экземпляре.

Точечные экземпляры лучше всего подходят для некритичных задач, где работу приложения можно прервать в любой момент и возобновить позже без существенного ущерба. К таким задачам можно отнести анализ данных, пакетную обработку и необязательные задачи.

Чтобы запустить точечный экземпляр, перейдите в панель управления EC2 и выберите в левом меню **Spot Requests** (Запросы на точечные экземпляры). В разделе **Instances** (Экземпляры) нажмите **Request Spot Instances** (Запросить точечные экземпляры), а затем следуйте той же процедуре, которая описывалась в подразделе «Экземпляры EC2 по требованию».

В этой главе мы не будем подробнее останавливаться на том, как работать с точечными экземплярами, потому что консоль EC2 предлагает доступные описания и помощь в работе с настройками. Дополнительную информацию можно найти на странице docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-spot-instances.html.

Как вы уже знаете, по умолчанию экземпляры EC2 работают на совместно используемом оборудовании: на одной физической или виртуальной машине могут размещаться экземпляры, которые принадлежат разным клиентам AWS. Но что, если вы хотите запускать свои экземпляры на отдельной платформе? Давайте рассмотрим выделенные экземпляры.

Выделенные экземпляры EC2

Некоторым предприятиям нужно, чтобы их приложения работали на выделенном оборудовании, не разделяя платформу ни с кем другим. Для этих целей AWS EC2 предоставляет **выделенные узлы** и **выделенные экземпляры**. Как и следовало ожидать, они стоят дороже, чем другие типы экземпляров. Так в каких же случаях они могут понадобиться?

Дело в том, что некоторые учреждения — особенно финансовые, медицинские и государственные — по закону обязаны выполнять строгие требования по обработке конфиденциальных данных или приобретать для запуска своих приложений лицензии, которые привязаны к аппаратному обеспечению.

Если вы используете выделенные экземпляры *без* выделенного узла, EC2 обеспечивает работу ваших приложений на отдельном гипервизоре, который предназначен исключительно для вас, но при этом не гарантирует фиксированного набора машин или оборудования. Другими словами, ваши экземпляры могут работать на разных физических узлах. А если в дополнение к выделенным экземплярам приобрести выделенный узел, то вам гарантируется монополярная среда с гипервизорами и узлами, в которой будут запускаться только ваши приложения, и вам не придется делить платформу с другими клиентами AWS.

Чтобы запустить выделенный экземпляр, выполните такие действия:

1. Начните с тех же шагов, что и при запуске экземпляра EC2 по требованию, о котором говорилось ранее в этой главе.
2. Теперь добавим еще один шаг. Разверните раздел **Advanced details** (Дополнительные сведения), прокрутите страницу до меню **Tenancy** (Режим аренды) и выберите в нем нужный вариант — в данном случае **Dedicated** — **run a dedicated instance** (Запустить выделенный экземпляр) (рис. 15.13).
3. Если вы хотите запустить выделенный экземпляр на выделенном узле, сначала нужно создать этот узел. Для этого на панели управления EC2 перейдите в раздел **Instances | Dedicated Hosts** (Экземпляры | Выделенные узлы) (рис. 15.14.).

Purchasing option Info
☐ Request Spot Instances

Capacity reservation Info

Select

Tenancy Info

Select

Select

Shared - run a shared hardware instance

Dedicated - run a dedicated Instance

Dedicated host - launch this Instance on a dedicated Host

Select

Рис. 15.13. Запуск выделенного экземпляра EC2

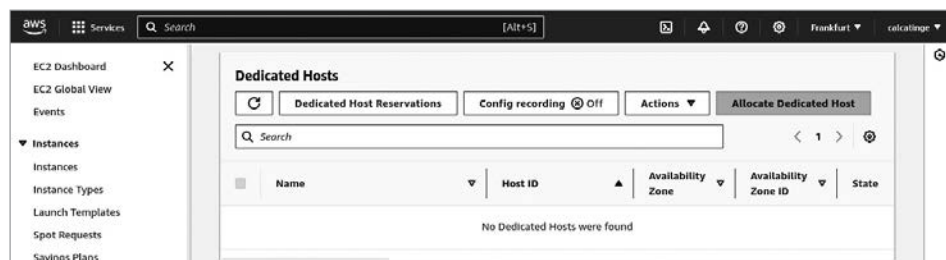


Рис. 15.14. Создание выделенного узла EC2

- Следуйте подсказкам мастера EC2, чтобы настроить выделенный узел, как вам нужно. После этого на нем можно запустить выделенный экземпляр: создайте экземпляр, как описано ранее, но на шаге 2 выберите в меню **Tenancy** вариант **Dedicated host — launch this instance on a dedicated Host** (Запустить этот экземпляр на выделенном узле).

Дополнительную информацию о выделенных узлах можно найти на странице docs.aws.amazon.com/AWSEC2/latest/UserGuide/dedicated-hosts-overview.html, а о выделенных экземплярах — на странице docs.aws.amazon.com/AWSEC2/latest/UserGuide/dedicated-instance.html.

Итак, мы рассмотрели типы экземпляров AWS EC2. Чтобы закрепить знания, обратитесь к официальной документации: docs.aws.amazon.com/AWSEC2/latest/UserGuide/Instances.html.

Мы еще не говорили о важных функциях EC2, которые позволяют оптимизировать ресурсы, когда вы разворачиваете и масштабируете экземпляры, — **образы машин Amazon (AMI) и группы размещения**. Подробнее узнать об AMI можно на странице docs.aws.amazon.com/AWSEC2/latest/UserGuide/AMIs.html, а в этой главе мы рассмотрим группы размещения, которые контролируют распределение ваших экземпляров по инфраструктуре EC2, чтобы обеспечить высокую доступность и оптимизировать рабочие нагрузки.

Группы размещения AWS EC2

Группы размещения позволяют указать, как ваши экземпляры располагаются на оборудовании или гипервизорах EC2, чтобы группировать или разделять экземпляры в зависимости от своих требований.

Группы размещения предоставляются бесплатно и делятся на три стратегии:

- **Кластерное размещение (cluster placement)**. Экземпляры размещаются в пределах одной зоны доступности (центра обработки данных), благодаря чему могут взаимодействовать между собой (но не с внешним миром) с минимальной задержкой и максимальным трафиком. Кластерное размещение особенно хорошо подойдет для приложений с высокопроизводительными вычислениями или репликацией данных, но, например, для веб-серверов оно будет не лучшим выбором.
- **Рассредоточенное размещение (spread placement)**. Когда вы запускаете несколько экземпляров EC2, всегда есть вероятность, что они будут работать на одной физической машине или гипервизоре. Это может породить единую точку отказа (например, аппаратное обеспечение) для ваших приложений. Рассредоточенное размещение обеспечивает аппаратную изоляцию между экземплярами. Другими словами, если вы запускаете несколько экземпляров в группе рассредоточенного размещения, можно гарантировать, что они будут работать на разных физических машинах, и единичный аппаратный сбой не сможет затронуть более одного экземпляра.
- **Размещение по разделам (partition placement)**. Группы размещения по разделам объединяют ваши экземпляры в логические формации (разделы) с аппаратной изоляцией между разделами, но не обязательно между экземплярами. Эту модель можно рассматривать как нечто среднее между кластерным и рассредоточенным размещением. Когда вы запускаете несколько экземпляров в рамках группы размещения по разделам, EC2 сделает все возможное, чтобы равномерно распределить экземпляры между разделами. Например, если у вас четыре раздела и 12 экземпляров, EC2 разместит по три экземпляра в каждом разделе. Таким образом, раздел выступает как вычислительная единица, которая состоит из нескольких экземпляров. EC2 поддерживает до семи экземпляров в одном логическом разделе.

Чтобы создать группу размещения, в левом меню панели управления EC2 перейдите в раздел **Network & Security | Placement Groups** (Сеть и безопасность | Группы

размещения) и нажмите **Create Placement Group** (Создать группу размещения). На следующем экране укажите имя и стратегию группы. Если нужно, добавьте теги (в формате «ключ — значение»), чтобы пометить или идентифицировать группу размещения. Когда все будет готово, нажмите **Create group** (Создать группу):

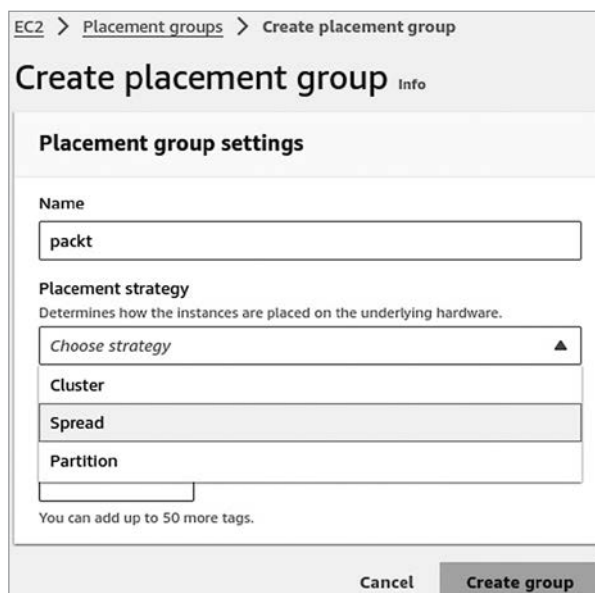


Рис. 15.15. Создание группы размещения

Дополнительную информацию о группах размещения EC2 можно найти на странице docs.aws.amazon.com/AWSEC2/latest/UserGuide/placement-groups.html.

Теперь, когда вы знакомы с различными типами экземпляров EC2, давайте посмотрим, как с ними работать.

Экземпляры AWS EC2 на практике

В этом разделе мы кратко рассмотрим некоторые ключевые операции и понятия, которые относятся к управлению экземплярами EC2. Начнем с жизненного цикла экземпляра.

Жизненный цикл экземпляра EC2

Когда вы используете экземпляры EC2 или управляете ими, важно понимать, из каких стадий складывается их жизненный цикл и как они переходят от одних стадий к другим: от запуска и выполнения до перехода в спящий режим или завершения работы. От того, в каком состоянии находится экземпляр, может зависеть тарификация и доступ к нему:

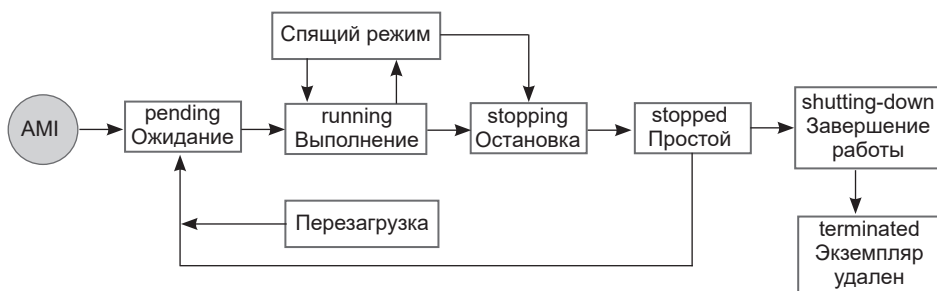


Рис. 15.16. Жизненный цикл экземпляра EC2

Давайте разберемся в этой схеме:

- В состоянии ожидания (**pending**) экземпляр загружается и инициализируется (в том числе после простоя).
- Экземпляр не всегда переходит из состояния ожидания в состояние выполнения (**running**): может потребоваться некоторое время, чтобы приложения, которые запущены в экземпляре, начали реагировать. EC2 списывает средства, когда экземпляр находится в состоянии выполнения (а также в состоянии остановки, если он переводится в спящий режим).
- В состоянии выполнения экземпляр в любой момент можно перезагрузить, если это необходимо. После перезагрузки экземпляр всегда продолжает выполняться на том же узле, в то время как при выходе из простоя EC2 может перевести его на другой узел.
- В состоянии простоя (**stopped**) плата за экземпляр не взимается, но вы продолжаете оплачивать любое дополнительное хранилище, которое подключено к экземпляру (кроме корневого тома).
- Чтобы временно отключить экземпляр, его можно остановить или перевести в спящий режим, который позволяет избежать потенциальной задержки при возвращении к работе. Если экземпляр больше не нужен, его можно удалить (перевести в состояние **terminated**), после чего плата за него больше не взимается. Если экземпляр удален, он может еще некоторое время отображаться на панели управления EC2, прежде чем окончательно утилизируется.

К экземпляру EC2 в состоянии выполнения можно подключиться с помощью SSH. В следующем разделе мы покажем, как это сделать.

Как подключиться к экземпляру AWS EC2

Как правило, каждый экземпляр EC2 служит для того, чтобы запускать определенное приложение или группу приложений. Чтобы администрировать и обслуживать соответствующую платформу, обычно нужен терминальный доступ. С точки зрения доступа AWS разделяет службы на две различные категории, похожие на те, что фигурируют в сетевых технологиях, — управляющий уровень

и уровень данных. Эти категории представляют собой способы доступа к экземплярам EC2:

- **Доступ к управляющему уровню.** Чтобы выполнять административные задачи на экземплярах EC2, можно использовать консоль AWS EC2 или терминал SSH.
- **Доступ к уровню данных.** Приложения, которые работают на экземплярах EC2, также могут открывать свои оконечные точки (порты) для связи с внешним миром. Чтобы управлять соответствующим сетевым трафиком, EC2 использует группы безопасности.

В этом разделе мы кратко рассмотрим доступ к обоим уровням. В частности, мы поговорим о том, как подключиться к экземпляру EC2 с помощью SSH и как передавать файлы с помощью SCP.

Как подключиться к экземпляру EC2 с помощью SSH

В режиме доступа к управляющему уровню можно подключиться к экземпляру EC2 по SSH и управлять им, как любой локальной машиной в сети. Вот как выглядит соответствующая команда SSH:

```
ssh -i ключ имя_пользователя@IP_экземпляра
```

Давайте подробнее разберем эту команду:

- *ключ* — это файл закрытого ключа, который мы создали, когда настраивали экземпляр, и сохранили в своей локальной системе (см. раздел «Экземпляры EC2 по требованию» выше).
- *имя_пользователя* — это учетная запись по умолчанию, которую EC2 назначила нашему экземпляру AMI Linux. Разным AMI могут соответствовать разные имена пользователей; у поставщика AMI можно уточнить имя пользователя для SSH по умолчанию. Например, в **Amazon Linux** учетная запись по умолчанию называется **ec2-user**, в **Ubuntu** — **ubuntu**, а в **Debian** — **admin**.
- *IP_экземпляра* — это публичный IP-адрес или доменное имя экземпляра EC2. Его можно найти на панели управления:



Рис. 15.17. Публичный IP-адрес и доменное имя экземпляра EC2

В нашем случае команда SSH выглядит так:

```
ssh -i packt_aws_key.pem ubuntu@3.68.98.173
```

Перед тем как подключаться, не забудьте установить нужные права доступа для закрытого ключа, чтобы он не был доступен для посторонних. Если этого не сделать, при попытке подключения возникнет ошибка.

```
chmod 400 packt_aws_key.pem
```

Так выглядит консоль нашего экземпляра EC2 после того, как мы успешно подключились по SSH:

```
alexandru@debian:~$ ssh -i packt_aws_key.pem ubuntu@3.68.98.173
The authenticity of host '3.68.98.173 (3.68.98.173)' can't be established.
ED25519 key fingerprint is SHA256:0Qpr2IsZyLf0vK9tLbikCGlbt5gU6qNI+S5ii6tcZUc.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '3.68.98.173' (ED25519) to the list of known hosts.
Welcome to Ubuntu 22.04.3 LTS (GNU/Linux 6.2.0-1017-aws x86_64)

* Documentation:  https://help.ubuntu.com
* Management:    https://landscape.canonical.com
* Support:        https://ubuntu.com/advantage
```

```
System information as of Tue Jan 16 09:35:00 UTC 2024
```

```
System load:  0.0           Processes:            98
Usage of /:   24.7% of 7.57GB Users logged in:       0
Memory usage: 23%          IPv4 address for eth0: 172.31.31.27
Swap usage:   0%
```

После этого с экземпляром EC2 можно взаимодействовать так же, как если бы он был обычным удаленным компьютером.

Далее пойдет речь о том, как обмениваться файлами с экземпляром EC2.

Как передавать файлы с помощью SCP

Чтобы обмениваться файлами с экземпляром EC2 в режиме доступа к уровню данных, нужно использовать утилиту `scp`, которая позволяет безопасно передавать файлы между сетевыми узлами по протоколу SCP.

Следующая команда копирует файл `README.md` с локального компьютера на удаленный экземпляр EC2. Обратите внимание, что ее нужно запускать с нашей машины, а не с экземпляра:

```
scp -i packt_aws_key.pem README.md ubuntu@3.68.98.173:~/
```

Файл копируется в домашний каталог пользователя `ubuntu (/home/ubuntu)` на экземпляре EC2. Чтобы перенести файл обратно, можно запустить такую команду:

```
scp -i packt_aws_key.pem ubuntu@3.68.98.173:~/README.md.
```

Обратите внимание, что команда `scp` вызывается подобно `ssh`, и файл закрытого ключа также передается с помощью аргумента `-i`.

Далее мы рассмотрим еще одно понятие, которое играет важную роль в управлении экземплярами EC2 и их масштабировании, — тома хранилища.

Тома хранилища EC2

Тома хранилища — это точки монтирования в экземпляре EC2, которые предоставляют дополнительное дисковое пространство (за дополнительную плату) — например, если вам нужно хранить большие файловые кэши или обширные журналы или если вы хотите подключить сетевое хранилище для критически важных данных, к которым совместно обращаются ваши экземпляры EC2. Тома хранилища EC2 ведут себя как *модульные жесткие диски*, которые можно монтировать и размонтировать в зависимости от ваших потребностей.

EC2 предоставляет тома хранилища двух типов:

- **Хранилище экземпляров.**
- **EBS** (Elastic Block Store).

Если вы умеете грамотно обращаться с томами хранилища, это поможет эффективно масштабировать приложения по мере их роста. Сначала рассмотрим тома хранилища экземпляров.

Тома хранилища экземпляров

Тома хранилища экземпляров — это диски, которые непосредственно (физически) подключены к экземпляру EC2. Следовательно, максимальная емкость и количество томов ограничены типом экземпляра. Например, к экземпляру *i3*, который оптимизирован для хранения данных, можно подключить до 8 накопителей SSD объемом до 1,9 Тбайт, а к экземпляру *m5d* общего назначения — не более четырех накопителей по 900 Гбайт. Дополнительные сведения о емкости экземпляров см. на странице aws.amazon.com/ec2/instance-types.

Том хранилища экземпляра предоставляется без дополнительной платы, если это корневой том с операционной системой, под управлением которой запускается экземпляр.

Не все типы экземпляров EC2 поддерживают тома хранилища. Например, типы общего назначения *t2* поддерживают только тома EBS. С другой стороны, именно тома EBS пригодятся, если вы захотите расширить объем хранилища за пределы максимальной емкости, которую поддерживает хранилище экземпляров.

Тома EBS

Данные на томах хранилища экземпляров сохраняются только внутри вашего экземпляра EC2. Если он будет остановлен или удален или если произойдет

авария, все данные пропадут. Чтобы сохранять критически важные данные экземпляров EC2, нужно использовать EBS.

Тома EBS — это гибкие и высокопроизводительные сетевые хранилища, которые служат как корневыми томами экземпляра, так и дополнительно монтируемыми томами. К каждому экземпляру EC2 можно подключить только один корневой том EBS, но дополнительных томов может быть несколько. Кроме того, с помощью функции Multi-Attach один и тот же том EBS можно одновременно прикрепить к нескольким экземплярам EC2. Подробнее об этой функции можно узнать на странице docs.aws.amazon.com/AWSEC2/latest/UserGuide/ebs-volumes-multi.html.

Когда вы создаете том EBS, он автоматически реплицируется в пределах зоны доступности вашего экземпляра, чтобы свести к минимуму задержки и потери данных. Если вы используете EBS, вам бесплатно доступен мониторинг состояния дисков и статистика Amazon CloudWatch в реальном времени. На томах EBS также можно размещать зашифрованные данные, что в последнее время часто требуется по нормативным стандартам хранения данных.

Тома хранилища EC2 опираются на инфраструктуру Amazon **S3** или **EFS**. Подробнее о различных вариантах хранилищ для экземпляров EC2 можно узнать на странице docs.aws.amazon.com/AWSEC2/latest/UserGuide/Storage.html.

Теперь давайте создадим и настроим том хранилища EBS и подключим его к нашему экземпляру EC2.

Как настроить том хранилища EBS

Вот что нужно сделать, чтобы создать и настроить том EBS:

1. На панели управления **EC2** откройте подраздел **Volumes** (Тома) из раздела **Elastic Block Store** в левой навигационной панели и нажмите на кнопку **Create Volume** (Создать том) в верхней части (рис. 15.18).
2. Укажите нужный тип, емкость и зону доступности тома — она должна совпадать с зоной, в которой находятся ваши экземпляры EC2. Если нужно восстановить том из резервной копии предыдущего экземпляра, можно указать идентификатор соответствующего моментального снимка.
3. Нажмите **Create Volume**. Если все прошло успешно, вы увидите сообщение о том, что том создан, а также получите его идентификатор.
4. Щелкните на идентификаторе тома или выберите этот том на левой панели навигации в том же подразделе **Volumes**. Нажмите кнопку **Actions** (Действия) и выберите **Attach Volume** (Прикрепить том) (рис. 15.19).
5. На следующем экране введите в поле **Instance** (Экземпляр) идентификатор экземпляра EC2 или тег имени, по которому его можно найти (рис. 15.20).
6. Нажмите кнопку **Attach Volume**. За считанные секунды EC2 инициализирует новый том EBS, а его состояние изменится на **in-use** (В эксплуатации). Устройство тома теперь готово, но чтобы его можно было использовать, надо отформатировать его под определенную файловую систему.

Instances

Instance Types

Launch Templates

Spot Requests

Savings Plans

Reserved Instances

Dedicated Hosts

Capacity Reservations New

▼ Images

AMIs

AMI Catalog

▼ Elastic Block Store

Volumes

Snapshots

Lifecycle Manager

▼ Network & Security

Security Groups

Elastic IPs

Placement Groups

Key Pairs

Network Interfaces

▼ Load Balancing

Load Balancers

Target Groups

Volume settings

Volume type

Info

General Purpose SSD (gp3)

① General Purpose SSD gp3 is now the default selection. gp3 provides up to 20% lower cost per GB than gp2.

[Learn More](#)

Size (GiB)

Info

1

Min: 1 GiB, Max: 16384 GiB. The value must be an integer.

IOPS

Info

3000

Min: 3000 IOPS, Max: 16000 IOPS. The value must be an integer.

Throughput (MiB/s)

Info

125

Min: 125 MiB, Max: 1000 MiB. Baseline: 125 MiB/s.

Availability Zone

Info

eu-central-1a

Snapshot ID - optional

Info

Don't create volume from a snapshot

Encryption

Info

Use Amazon EBS encryption as an encryption solution for your EBS resources associated with your EC2 instances.

☐ Encrypt this volume

Рис. 15.18. Создание тома EBS

vol-061fe67366d2c0456					Actions ▲	Delete	Modify
					Create snapshot		
					Attach volume		
					Detach volume		
					Force detach volume		
					Manage auto-enabled I/O		
Volume ID	Size	Type					
vol-061fe67366d2c0456	1 GiB	gp3					
AWS Compute Optimizer finding	Volume state	IOPS					
Opt-In to AWS Compute Optimizer for recommendations. Learn more	Available	3000					
Encryption	KMS key ID	KMS key alias					
Not encrypted	-	-					
Fast snapshot restored	Snapshot	Availability Zone					
No	-	eu-central-1a					
Multi-Attach enabled	Attached resources	Outposts ARN					
No	-	-					
			KMS key ARN				
			-				
			Created				
			Tue Jan 16 2024 18:48:27 GMT+0200 (Eastern European Standard Time)				

Рис. 15.19. Прикрепление тома EBS к экземпляру EC2

7. Давайте подключимся по SSH к экземпляру EC2, к которому мы прикрепили том:

```
ssh -i packt_aws_key.pem ubuntu@3.68.98.173
```

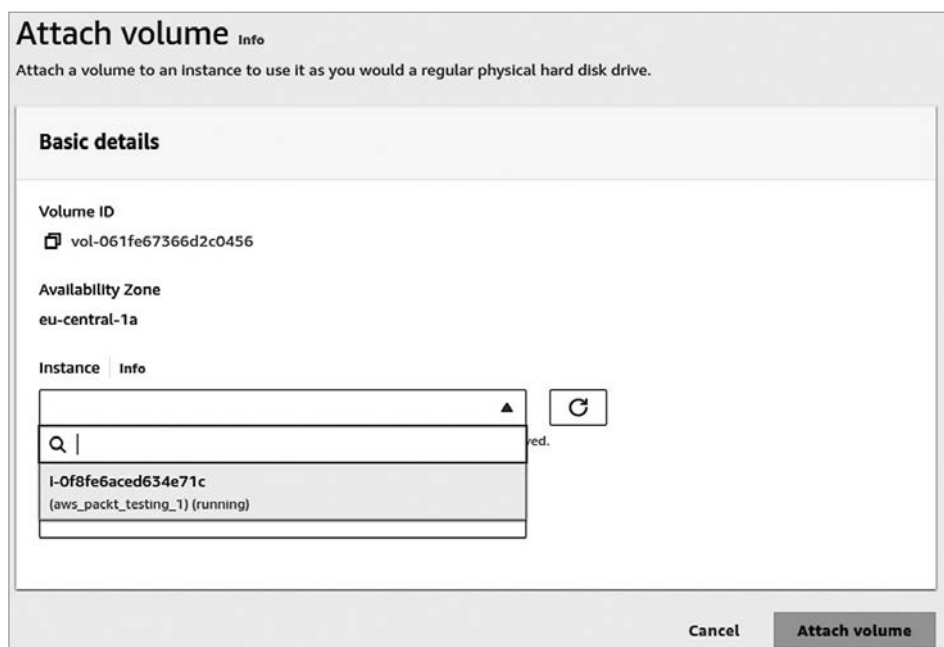


Рис. 15.20. Ввод идентификатора экземпляра EC2 для прикрепления тома

8. Далее имеет смысл просмотреть сведения о дисках, которые доступны в нашем экземпляре EC2. Это можно сделать командой `lsblk`:

```
ubuntu@ip-172-31-31-27:~$ lsblk
NAME        MAJ:MIN RM  SIZE RO TYPE MOUNTPOINTS
loop0        7:0      0   24.9M  1 loop /snap/amazon-ssm-agent/7628
loop1        7:1      0   55.7M  1 loop /snap/core18/2812
loop2        7:2      0   63.5M  1 loop /snap/core20/2015
loop3        7:3      0  111.9M  1 loop /snap/lxd/24322
loop4        7:4      0   40.9M  1 loop /snap/snapd/20290
loop5        7:5      0   63.9M  1 loop /snap/core20/2105
loop6        7:6      0   40.4M  1 loop /snap/snapd/20671
xvda        202:0     0     8G   0 disk
├─xvda1     202:1     0    7.9G   0 part /
├─xvda14    202:14    0     4M   0 part
└─xvda15    202:15    0   106M   0 part /boot/efi
xvdf        202:80    0     1G   0 disk
```

Глядя на размер томов, можно сразу определить, какой из них мы только что добавили — это `xvdf` объемом 1 Гбайт. Другой том (`xvda`) — это оригинальный корневой том нашего экземпляра `t3.micro`.

9. Теперь давайте проверим, есть ли на нашем новом томе EBS файловая система:

```
sudo file -s /dev/xvdf
```


Данная команда выводит `/dev/xvdf: data`, это означает, что у тома еще нет файловой системы.

10. Создайте файловую систему с помощью утилиты командной строки `mkfs`:

```
sudo mkfs -t xfs /dev/xvdf
```

Здесь мы с помощью ключа `-t (--type)` задали тип файловой системы XFS. Это высокопроизводительная журналируемая система, которую поддерживает большинство дистрибутивов Linux, причем в некоторых из них она установлена по умолчанию.

11. Если теперь проверить файловую систему, то вместо `data` мы увидим подробную информацию о ней и таким образом убедимся, что диск отформатирован:

```
ubuntu@ip-172-31-31-27:~$ sudo file -s /dev/xvdf
/dev/xvdf: SGI XFS filesystem data (blksz 4096, inosz 512, v2 dirs)
```

12. Теперь давайте добьемся, чтобы том стал доступен для нашей локальной файловой системы. Для этого создадим в корневом каталоге точку монтирования `packt_drive`:

```
sudo mkdir /packt_drive
sudo mount /dev/xvdf /packt_drive
```

Теперь том EBS смонтирован, и, если обратиться к каталогу `/packt_drive`, мы на самом деле обратимся к тому EBS:

```
ubuntu@ip-172-31-31-27:~$ ls -la packt_drive/
total 4
drwxr-xr-x  2 root root   23 Jan 16 17:03
drwxr-xr-x 20 root root 4096 Jan 16 17:02
-rw-r--r--  1 root root    0 Jan 16 17:03 README.md
```

С томами EBS можно работать практически так же, как с любыми томами в Linux, и здесь вам пригодятся знания, которые вы получили из главы 6 «Как работать с дисками и файловыми системами». Теперь давайте рассмотрим, как отсоединить том EBS.

Как отсоединить том EBS

Перед тем как отсоединить том EBS, его нужно размонтировать. В нашем случае, поскольку мы уже находимся в терминале, из которого есть доступ к тому, давайте размонтируем его с помощью такой команды:

```
sudo umount -d /dev/xvdf
```

После этого можно зайти в панель управления EC2, перейти в раздел **Volumes**, выбрать нужный том из тех, которые находятся в эксплуатации (*in-use*), открыть меню **Actions** в правом верхнем углу и выбрать **Detach Volume** (Отсоединить том). Подтвердите операцию и дождитесь, пока том отсоединится.

На этом мы закончим обзор консоли AWS EC2 и соответствующих операций управления. Чтобы получить исчерпывающую информацию о EC2, обратитесь к официальной документации: docs.aws.amazon.com/ec2.

В административных задачах EC2 мы до сих пор использовали только веб-консоль AWS. Если нужно автоматизировать управление экземплярами EC2, вам может пригодиться интерфейс командной строки, который мы рассмотрим далее.

Интерфейс командной строки AWS CLI

Интерфейс командной строки AWS (AWS CLI) — это унифицированный инструмент, который позволяет управлять ресурсами и службами AWS из сеанса терминала на локальном компьютере. В отличие от веб-консоли AWS, которая предоставляет графический интерфейс, в AWS CLI все функциональные возможности обеспечиваются внутри терминала на вашей машине. AWS CLI доступен для всех основных операционных систем — Linux, macOS и Windows. В следующем разделе мы продемонстрируем, как установить его на локальную машину Linux.

Как установить AWS CLI

На момент написания этой книги последний выпуск AWS CLI — версия 2. На странице docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html вы найдете подробные инструкции о том, как установить AWS CLI не только в Linux, но и в остальных операционных системах. Для примеров в этом разделе мы будем использовать компьютер под управлением Debian.

Итак, чтобы установить AWS CLI, выполните такие действия:

1. Сначала загрузите пакет AWS CLI ([awscliv2.zip](#)):

```
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip"
-o "awscliv2.zip"
```

2. Распакуйте архив и запустите установочный сценарий:

```
unzip awscliv2.zip
sudo ./aws/install
```

3. Теперь в системе должна быть установлена утилита `aws`. Давайте проверим ее версию:

```
alexandru@debian:~$ aws --version
aws-cli/2.25.11 Python/3.12.9 Linux/6.1.0-32-amd64 exe/x86_64.debian.12
```

4. Чтобы начать знакомство с интерфейсом командной строки AWS, можно вызвать встроенную справку с помощью такой команды:

```
aws help
```

Перед тем как управлять ресурсами AWS EC2 с помощью `aws`, сначала нужно настроить локальное окружение, чтобы установить необходимое доверие с окончной точкой AWS. Давайте этим займемся.

Как настроить AWS CLI

Чтобы настроить локальное окружение AWS на своем компьютере, вам понадобится установить ключ доступа к AWS. Если он у вас уже есть, эту часть можно пропустить.

1. Конфигурация AWS CLI запрашивает идентификатор и значение ключа доступа (**AWS Access Key ID** и **AWS Secret Access Key**). Эти данные можно получить в веб-консоли EC2, как описано в следующем пункте.
2. В правом верхнем углу консоли откройте выпадающий список рядом с именем вашей учетной записи и выберите пункт **Security Credentials** (Реквизиты безопасности). Если вы еще не сгенерировали ключ доступа, перейдите на вкладку **Access keys** (Ключи доступа) и нажмите кнопку **Create access key** (Создать ключ). Сохраните идентификатор и сам ключ доступа в надежном месте, чтобы использовать их в дальнейшем.
3. Теперь, когда у вас есть ключ, можно настроить среду AWS на локальной машине. Для этого запустите такую команду:

```
aws configure
```

4. Эта команда запросит у вас несколько значений:

```
alexandru@debian:~$ aws configure
AWS Access Key ID [None]: AKIAIOSFODNN7EXAMPLE
AWS Secret Access Key [None]: je7MtGbClwBF/2Zp9Utk/h3yCo8nvbEXAMPLEKEY
Default region name [None]: eu-central-1
Default output format [None]:
```

5. Мы задали для **региона по умолчанию** значение **eu-central-1** (Германия, Франкфурт). Если этот регион вам не подходит, введите свой регион или оставьте значение по умолчанию (**None**). Если вы не настроили регион по умолчанию, вам придется указывать регион каждый раз, когда вы вызываете команду **aws**.

Теперь мы готовы к тому, чтобы использовать AWS CLI. Давайте для начала посмотрим список наших экземпляров EC2.

Как запрашивать сведения об экземплярах EC2

Следующая команда предоставляет подробную информацию об экземплярах EC2:

```
aws ec2 describe-instances
```

В таком виде эта команда выдает объемный вывод в формате JSON (слишком длинный, чтобы представить его здесь) с подробной информацией обо всех экземплярах EC2, которые принадлежат нам и находятся в нашем регионе по умолчанию (**eu-central-1**). Регион также можно указать вручную с помощью ключа **--region**:

```
aws ec2 describe-instances --region eu-central-1
```

Ключ `--filters` позволяет сузить выборку и перечислить только те экземпляры EC2, которые соответствуют определенному тегу — например, тегу `env: packt`, которым мы ранее поместили наши экземпляры:

```
aws ec2 describe-instances \
  --filters "Name=tag-key,Values=env" \
  --filters "Name=tag-value,Values=packt"
```

Здесь первый аргумент `--filters` задает ключ (`env`), а второй — значение (`packt`).

Если комбинировать команду `aws` с утилитой `jq`, которая позволяет обрабатывать данные в формате JSON, из выборки можно извлечь нужные поля. Например, следующая команда выводит поля `InstanceId`, `ImageId` и `BlockDeviceMappings` для экземпляров EC2, которые помечены тегом `env:packt`:

```
alexandru@debian:~$ aws ec2 describe-instances \
  --filters "Name=tag-key,Values=env" \
  --filters "Name=tag-value,Values=packt" | \
  jq '.Reservations[].Instances[] | { InstanceId, ImageId, BlockDeviceMappings
}'
{
  "InstanceId": "i-0f8fe6aced634e71c",
  "ImageId": "ami-faab6bdbac9486fb",
  "BlockDeviceMappings": [
    {
      "DeviceName": "/dev/sda1",
      "Ebs": {
        "AttachTime": "2024-01-14T13:25:46+00:00",
        "DeleteOnTermination": true,
        "Status": "attached",
        "VolumeId": "vol-03d55241cf12c6772"
      }
    }
  ]
}
```

Обратите внимание на свойство `DeviceName`, в котором отражено только одно блочное устройство (`/dev/sda1`), потому что мы отсоединили том EBS, который подключали до этого.

Вывод команды `aws ec2 describe-instances` можно фильтровать по любому свойству. Например, следующая команда отображает экземпляры EC2 с определенным идентификатором АМИ (`ImageId`):

```
aws ec2 describe-instances \
  --filters "Name=image-id,Values=ami-0e999cbd62129e3b1"
```

Обратите внимание: чтобы использовать имя свойства в фильтре, его надо преобразовать из «верблюжьего» регистра (`ImageId`) в дефисную нотацию (`image-id`). Не забывайте об этом, когда составляете запросы с фильтрами.

Теперь давайте посмотрим, как из командной строки создать новый экземпляр EC2 с тем же типом АМІ и с той же группой безопасности, что и у нашей текущей машины.

Как создать экземпляр EC2

Чтобы создать новый экземпляр EC2, нам понадобятся сведения о группе безопасности, внутри которой мы его создаем. Следующая команда выводит группы безопасности текущего экземпляра (i-0f8fe6aced634e71c):

```
alexandru@debian:~$ aws ec2 describe-instances \
  --filters "Name=instance-id,Values=i-0f8fe6aced634e71c" \
  --query "Reservations[].Instances[].SecurityGroups[]"
[
  {
    "GroupName": "launch-wizard-5",
    "GroupId": "sg-0aa7c8ef75503a9aa"
  }
]
```

Чтобы получить идентификатор группы (GroupId) напрямую, можно запустить такую команду:

```
aws ec2 describe-instances \
  --filters "Name=instance-id,Values=i-0f8fe6aced634e71c" \
  --query "Reservations[].Instances[].SecurityGroups[].GroupId"
```

Здесь мы использовали ключ `--query`, чтобы указать точный путь к полю `GroupId`. Этот ключ в некоторой степени похож на команду `jq`, но у него гораздо меньше возможностей.

Помимо идентификатора группы, нам также понадобится идентификатор образа АМІ. Чтобы получить его, откройте консоль AWS EC2 и выберите на левой панели **Images | AMI Catalog** (Образы | Каталог АМІ). Найдите нужную машину (в нашем случае это **Ubuntu Server 22.04 LTS**) и скопируйте идентификатор АМІ для 64-битной версии (ami-0faab6bdbac9486fb).

Чтобы запустить новый экземпляр с заданной АМІ и идентификатором группы безопасности, используйте команду `aws ec2 run-instances`:

```
aws ec2 run-instances \
  --image-id ami-0faab6bdbac9486fb \
  --count 1 \
  --instance-type t3.micro \
  --key-name packt_aws_key \
  --security-group-ids sg-0aa7c8ef75503a9aa \
  --placement AvailabilityZone=eu-central-1b
```

Вот что означают ключи этой команды:

- **image-id**: идентификатор образа АМІ. Мы используем тот же тип АМІ (*Ubuntu Linux*), что и в экземпляре, который ранее создали в веб-консоли AWS EC2;

- `count`: количество экземпляров;
- `instance-type`: тип экземпляра EC2;
- `key-name`: имя файла закрытого ключа SSH, с помощью которого мы будем подключаться к нашему новому экземпляру. Мы используем тот же ключ, который создали ранее в веб-консоли AWS;
- `security-group-ids`: группы безопасности, с которыми связан экземпляр. Мы указали ту же группу, к которой относится наш текущий экземпляр;
- `placement`: зона доступности, в которой размещается наш экземпляр.

Если эта команда успешно выполнится, вы увидите в консоли новый экземпляр EC2 в дополнение к тому, который мы создали раньше:



Рис. 15.21. Новый экземпляр в консоли AWS

Как видите, у свежесозданного экземпляра нет имени. Давайте присвоим ему имя и добавим несколько тегов с помощью командной строки.

Как редактировать имя и теги экземпляра EC2

Следующая команда присваивает нашему новому экземпляру имя `aws_packt_testing_2`:

```
aws ec2 create-tags --resources i-091e2f515d15c3b0b \
  --tags Key=Name,Value=aws_packt_testing_2
```

Теперь давайте добавим тег. Имя и теги можно задать одной командой, но мы решили использовать две разные команды, чтобы пример был понятнее. Вот как добавить тег:

```
aws ec2 create-tags \
  --resources i-0e1692c9dfdf07a8d \
  --tags Key=env,Value=packt
```

Чтобы убедиться, что экземпляр фильтруется по заданному имени и тегу, можно запустить такую команду:

```
alexandru@debian:~$ aws ec2 describe-instances \
  --filters "Name=tag-key,Values=env" \
  --filters "Name=tag-value,Values=packt" \
  --query "Reservations[0].Instances[0].InstanceId"
[
```

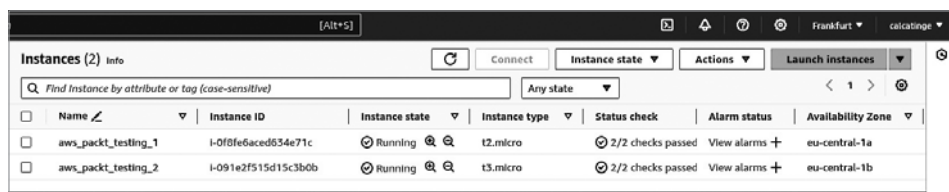
```

    "i-0f8fe6aced634e71c",
    "i-091e2f515d15c3b0b"
]

```

Здесь видно, что у двух экземпляров EC2 есть одинаковый тег `env: packt`.

В консоли EC2 вы по-прежнему увидите два экземпляра с этими же идентификаторами и убедитесь, что теперь у второго экземпляра отображается его новое имя (`aws_packt_testing_2`):



	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone
☐	aws_packt_testing_1	i-0f8fe6aced634e71c	Running	t2.micro	2/2 checks passed	View alarms +	eu-central-1a
☐	aws_packt_testing_2	i-091e2f515d15c3b0b	Running	t3.micro	2/2 checks passed	View alarms +	eu-central-1b

Рис. 15.22. Отображение экземпляров в консоли EC2

Далее давайте посмотрим, как завершить работу экземпляра EC2 и удалить его с помощью командной строки.

Как удалить экземпляр EC2

Чтобы завершить работу экземпляра и удалить его, можно запустить команду `aws ec2 terminate-instance`. Если нужно не удалять экземпляр, а приостановить его на время, используйте команду `aws ec2 stop-instances`.

Следующая команда удаляет экземпляр с идентификатором `i-091e2f515d15c3b0b`. В выводе указано, что экземпляр переходит из состояния выполнения (`running`) в состояние завершения работы (`shutting-down`):

```

alexandru@debian:~$ aws ec2 terminate-instances --instance-ids
i-091e2f515d15c3b0b
{
  "TerminatingInstances": [
    {
      "CurrentState": {
        "Code": 32,
        "Name": "shutting-down"
      },
      "InstanceId": "i-091e2f515d15c3b0b",
      "PreviousState": {
        "Code": 16,
        "Name": "running"
      }
    }
  ]
}

```

В конце концов экземпляр перейдет в состояние удаленного (`terminated`) и больше не будет отображаться в консоли AWS EC2. В интерфейсе командной стро-

ки AWS он останется на некоторое время, пока EC2 окончательно его не утилизирует (обычно это происходит в течение часа после того, как вы удалили экземпляр). Когда вы отправляете запросы и выполняете административные операции в AWS CLI, всегда рекомендуется отфильтровывать экземпляры, которые находятся в состоянии `terminated` или `shutting-down`. Вот как в консоли EC2 выглядит экземпляр, который находится в состоянии удаленного:

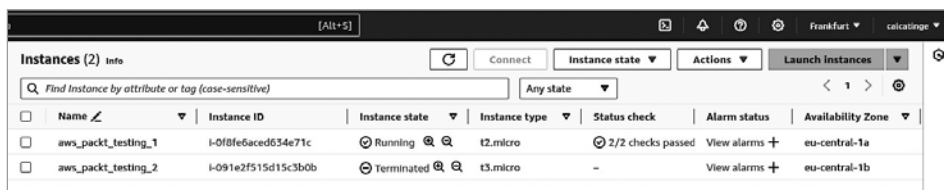


Рис. 15.23. Удаленный экземпляр в консоли EC2

На этом мы завершаем обзор AWS EC2. Обратите внимание, что мы рассмотрели только самые элементарные понятия и операции, которые относятся к управлению облачной средой в AWS.

В этом разделе вы получили общее представление об облачных ресурсах AWS EC2, и это поможет вам принимать эффективные решения, когда понадобится управлять соответствующими рабочими нагрузками. Для опытных пользователей примеры AWS CLI станут хорошей отправной точкой для того, чтобы автоматизировать рабочие процессы по управлению облачной средой EC2.

Теперь давайте перейдем к следующему облачному решению — Microsoft Azure.

Microsoft Azure

Microsoft Azure — это публичная облачная инфраструктура, которая позволяет создавать и разворачивать службы приложений. Azure предоставляет полноценное хорошо масштабируемое решение **IaaS** по относительно невысокой цене, что позволяет охватить широкий круг пользователей с разными бизнес-требованиями — от небольших команд до крупных коммерческих предприятий, включая финансовые, медицинские и правительственные учреждения.

В этом разделе мы рассмотрим основные рабочие процессы развертывания с помощью Azure:

- Как создать виртуальную машину Linux.
- Как регулировать мощность виртуальных машин.
- Как добавить дополнительное хранилище к виртуальной машине.
- Как работать с интерфейсом командной строки Azure.

После того как вы создадите бесплатную учетную запись Azure, зайдите на портал Azure (portal.azure.com). Для быстрого и удобного доступа к ресурсам

можно закрепить навигационное меню портала в левой части экрана: для этого перейдите к настройкам портала (значок шестеренки в правом верхнем углу) и выберите для меню портала режим по умолчанию **Docked** (Закрепленный). Мы будем использовать этот режим во всех снимках экрана в этой главе.

Теперь давайте создадим в Azure наш первый ресурс — виртуальную машину Ubuntu.

Как создать и развернуть виртуальную машину

Мы будем следовать пошаговой процедуре, которую предлагает мастер создания ресурсов на портале Azure. Для начала создадим вычислительный ресурс для виртуальной машины.

1. **Создайте вычислительный ресурс.** Нажмите **Create a resource** (Создать ресурс) в левом навигационном меню или в разделе **Azure services** (Службы Azure) в главном окне. Вы перейдете в магазин Azure Marketplace, где можно найти нужный ресурс по ключевым словам или отфильтровать выборку по типу ресурса. Давайте ее сузим, выбрав **Compute** (Вычислительные ресурсы), а затем **Ubuntu Server 22.04 LTS** в меню **Popular Marketplace products** (Популярные продукты Marketplace). Чтобы ознакомиться с подробным описанием образа, можно щелкнуть на **Learn more** (Узнать больше), а затем нажать кнопку **Create** (Создать).

После того как вы выберете **Ubuntu Server 22.04 LTS**, вы перейдете в интерфейс, который позволяет оснастить и создать новую виртуальную машину. Этот интерфейс выглядит так (рис. 15.24).

На вкладке **Basics** (Основные настройки) представлены такие сведения, как тип подписки, группа ресурсов, регион, образ экземпляра и его архитектура. На других вкладках можно настроить прочие свойства ресурса: **Disks** (Диски), **Networking** (Сеть), **Management** (Управление), **Monitoring** (Мониторинг), **Advanced** (Расширенные настройки) и **Tags** (Теги).

2. **Настройте группу ресурсов.** Прежде всего нужно задать значения для параметров **Subscription** (Подписка) и **Resource group** (Группа ресурсов). Группа ресурсов Azure — это набор активов, которые связаны с конкретным развертыванием: хранилище, сетевые интерфейсы, группы безопасности и т. д. Поскольку это наша первая виртуальная машина, мы создадим для нее новую группу ресурсов, которую назовем **packt-demo**. Если бы мы уже раньше создали группу ресурсов, ее можно было бы указать здесь.
3. **Настройте сведения об экземпляре.** На этом этапе мы зададим различные свойства экземпляра, такие как имя виртуальной машины, регион и размер. Мы назовем виртуальную машину **packt-ubuntu-demo** и поместим ее в регион (Europe) **France Central**, который ближе всего к нашему географическому местоположению. От размера машины будут напрямую зависеть наши расходы. На следующем снимке экрана показаны параметры, которые мы настроили (рис. 15.25).

Home > Create a resource >

Create a virtual machine

Basics Disks Networking Management Monitoring Advanced Tags Review + create

Create a virtual machine that runs Linux or Windows. Select an image from Azure marketplace or use your own customized image. Complete the Basics tab then Review + create to provision a virtual machine with default parameters or review each tab for full customization. Learn more

This subscription may not be eligible to deploy VMs of certain sizes in certain regions.

Project details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription *

Resource group *
Create new

Instance details

Virtual machine name *

Region *

Availability options

Availability zone *

☒ You can now select multiple zones. Selecting multiple zones will create one VM per zone. Learn more

Security type
Configure security features

Image *
See all images | Configure VM generation

VM architecture ☐ Arm64 ☒ x64

Рис. 15.24. Создание виртуальной машины в Azure

Чтобы просмотреть доступные варианты образов и размеров, можно щелкнуть соответственно на *See all images* (Все образы) и *See all sizes* (Все размеры). Azure также предоставляет онлайн-калькулятор стоимости различных ресурсов: azure.microsoft.com/pricing/calculator.

4. **Настройте доступ по SSH.** Мы включим доступ по SSH с аутентификацией по открытому ключу, указав в качестве имени пользователя `packt`, а в качестве имени ключа — `packt-ubuntu-demo_key`. Кроме этого, нужно задать правила для входящих портов нашего экземпляра, чтобы разрешить доступ по SSH. А если, например, на нашей машине будет работать веб-сервер, то надо будет также включить доступ по протоколам HTTP и HTTPS (рис. 15.26).

Теперь все готово для того, чтобы создать виртуальную машину. Мастер может провести вас через дополнительные шаги, на которых можно настроить диски и сетевую конфигурацию экземпляра, но в этом упражнении мы оставим эти настройки без изменений.

Project details
Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * ⓘ Free Trial ✓

Resource group * ⓘ (New) packt-demo ✓
[Create new](#)

Instance details

Virtual machine name * ⓘ packt-ubuntu-demo ✓

Region * ⓘ (Europe) France Central ✓

Availability options ⓘ No infrastructure redundancy required ✓

Security type ⓘ Trusted launch virtual machines ✓
[Configure security features](#)

Image * ⓘ Ubuntu Server 22.04 LTS - x64 Gen2 ✓
[See all images](#) | [Configure VM generation](#)

VM architecture ⓘ ☐ Arm64 ☒ x64

Run with Azure Spot discount ⓘ ☐

Size * ⓘ Standard_B1s - 1 vcpu, 1 GiB memory (€8.06/month) (free services eligible) ✓
[See all sizes](#)

Information: You are in the free trial period. Costs associated with this VM can be covered by any remaining credits on your subscription. [Learn more](#) ⓘ

Рис. 15.25. Настройка виртуальной машины Azure

Administrator account

Authentication type ⓘ ☒ SSH public key ☐ Password

Information: Azure now automatically generates an SSH key pair for you and allows you to store it for future use. It is a fast, simple, and secure way to connect to your virtual machine.

Username * ⓘ packt ✓

SSH public key source Generate new key pair ✓

Key pair name * packt-ubuntu-demo_key ✓

Inbound port rules
Select which virtual machine network ports are accessible from the public internet. You can specify more limited or granular network access on the Networking tab.

Public inbound ports * ⓘ ☐ None ☒ Allow selected ports

Select inbound ports * SSH (22) ✓

Information: All traffic from the internet will be blocked by default. You will be able to change inbound port rules in the VM > Networking page.

Рис. 15.26. Настройка доступа по SSH к виртуальной машине

5. **Проверьте и разверните виртуальную машину.** Щелкните на **Review + create** (Проверить и создать), чтобы мастер развертывания проверил конфигурацию виртуальной машины. Если все получится, через несколько минут вы увидите сообщение **Validation passed** (Проверка прошла успешно), в котором будет указана информация о продукте и почасовой тариф вашего экземпляра. Нажав кнопку **Create**, вы подтвердите, что согласны с соответствующими условиями использования, и виртуальная машина будет развернута за несколько минут (рис. 15.27).

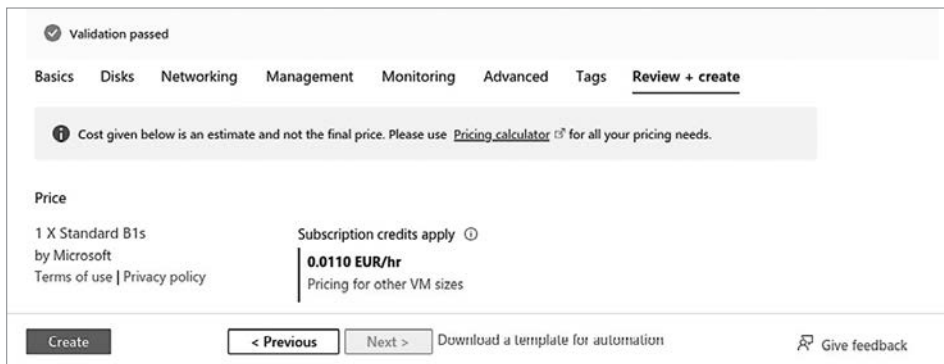


Рис. 15.27. Создание виртуальной машины

В процессе мастер предложит вам загрузить закрытый ключ SSH для доступа к экземпляру:

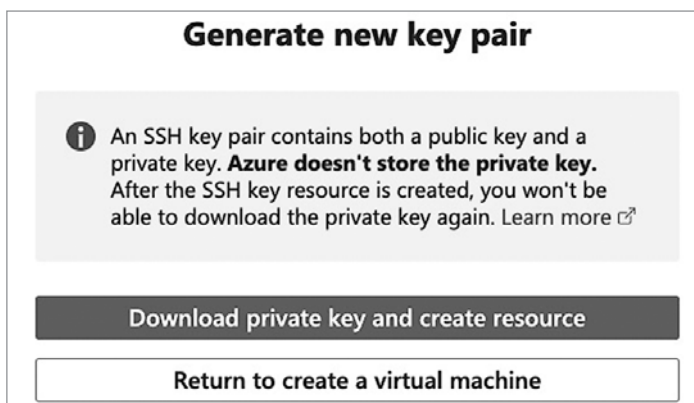


Рис. 15.28. Загрузка закрытого ключа SSH для доступа к виртуальной машине

6. **Завершите развертывание.** Если развертывание завершится без проблем, вы увидите всплывающее сообщение **Deployment succeeded** (Развертывание прошло успешно) и кнопку **Go to resource** (Перейти к ресурсу), которая приведет вас к новой виртуальной машине (рис. 15.29).

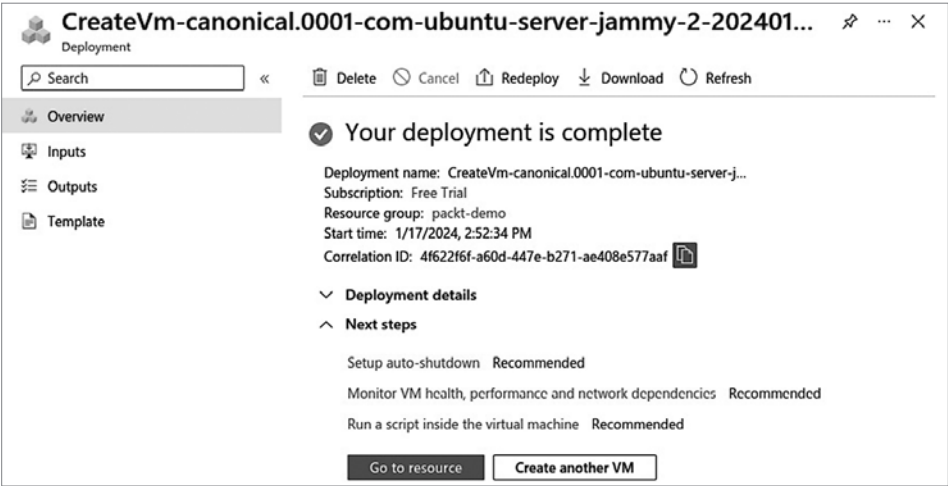


Рис. 15.29. Успешное развертывание виртуальной машины

Если вас интересуют подробности развертывания, щелкните на заголовке раскрывающейся панели **Deployment details** (Детали развертывания):

Deployment details			
Resource	Type	Status	Operation detail:
✓ packt-ubuntu-demo	Microsoft.Compute/vir...	OK	Operation details
✓ packt-ubuntu-demo348	Microsoft.Network/net...	Created	Operation details
✓ packt-ubuntu-demo-nsg	Microsoft.Network/net...	OK	Operation details
✓ packt-ubuntu-demo-ip	Microsoft.Network/pu...	OK	Operation details
✓ packt-ubuntu-demo-vnet	Microsoft.Network/virt...	OK	Operation details

Рис. 15.30. Детали развертывания

Давайте перечислим ресурсы, которые были созданы при развертывании виртуальной машины:

packt-ubuntu-demo	Узел виртуальной машины
packt-ubuntu-demo348	Сетевой интерфейс виртуальной машины
packt-ubuntu-demo-ip	IP-адрес виртуальной машины
packt-ubuntu-demo-vnet	Виртуальная сеть, связанная с группой ресурсов packt-demo
packt-ubuntu-demo-nsg	Группа сетевой безопасности, которая контролирует входящий и исходящий доступ для нашего экземпляра

Для каждой виртуальной машины Azure создает новый набор типов ресурсов, которые упоминались раньше, за исключением виртуальной сети, которая соответствует группе ресурсов, если экземпляр размещается в существующей группе. Не забывайте, что мы ранее создали новую группу ресурсов `packt-demo`, которая не отображается в отчете о развертывании.

Давайте попробуем подключиться к нашему свеже созданному экземпляру `packt-ubuntu-demo`. На левой навигационной панели перейдите в раздел **Virtual Machines** (Виртуальные машины) и выберите указанный экземпляр. На вкладке **Overview** (Обзор) вы увидите основные сведения о виртуальной машине, включая ее публичный IP-адрес (в нашем случае `20.19.173.232`).

Теперь, когда мы развернули виртуальную машину, нужно убедиться, что к ней можно подключиться по SSH.

Как подключиться к виртуальной машине с помощью SSH

Как обычно, прежде чем подключаться к виртуальной машине по SSH, установите нужные права доступа к файлу закрытого ключа:

```
chmod 400 packt-ubuntu-demo_key.pem
```

Затем подключитесь к экземпляру Ubuntu с помощью такой команды:

```
ssh -i packt-ubuntu-demo_key.pem packt@20.19.173.232
```

Здесь нужно использовать закрытый ключ SSH (`packt-ubuntu-demo_key.pem`) и учетную запись администратора, с которой связан экземпляр (в нашем случае `packt`). Можно также щелкнуть на **Connect** (Подключиться) на вкладке **Overview** виртуальной машины, а затем щелкнуть на **SSH**. В результате появится окно, в котором вы сможете увидеть предыдущие команды, чтобы скопировать и вставить их в терминал.

```
alexandru@debian:~$ chmod 400 packt-ubuntu-demo_key.pem
alexandru@debian:~$ ssh -i packt-ubuntu-demo_key.pem packt@20.19.173.232
The authenticity of host '20.19.173.232 (20.19.173.232)' can't be established.
ED25519 key fingerprint is SHA256:BlbczCJYUdEf5+aMdYlVow/QC9zLATBFvJL9G/DtbrM.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '20.19.173.232' (ED25519) to the list of known hosts.
Welcome to Ubuntu 22.04.3 LTS (GNU/Linux 6.2.0-1018-azure x86_64)
```

Рис. 15.31. Подключение к виртуальной машине Azure

Теперь, когда мы создали нашу первую виртуальную машину в Azure и подключились к ней, давайте рассмотрим наиболее распространенные операции, с помощью которых можно управлять виртуальными машинами.

Как управлять виртуальными машинами

По мере того как ваши приложения развиваются, им требуются все более мощные и производительные виртуальные машины. Вам как системным администраторам стоит представлять себе, как потребляются облачные ресурсы. Azure предоставляет необходимые инструменты для того, чтобы отслеживать работоспособность и производительность виртуальных машин; эти инструменты доступны на вкладке **Monitoring** (Мониторинг) на странице управления виртуальной машиной.

Если у вас экономичная машина, у которой мало виртуальных процессоров и памяти, производительность вашего приложения может пострадать. С другой стороны, чересчур мощный экземпляр приведет к неоправданным расходам. Давайте посмотрим, как изменить размер виртуальной машины в Azure.

Как изменить размер виртуальной машины

Azure позволяет довольно легко изменить размер виртуальной машины. На портале перейдите в раздел **Virtual Machines**, выберите свой экземпляр и откройте панель **Size** (Размер) в разделе **Availability + Scale** (Доступность и масштабирование):

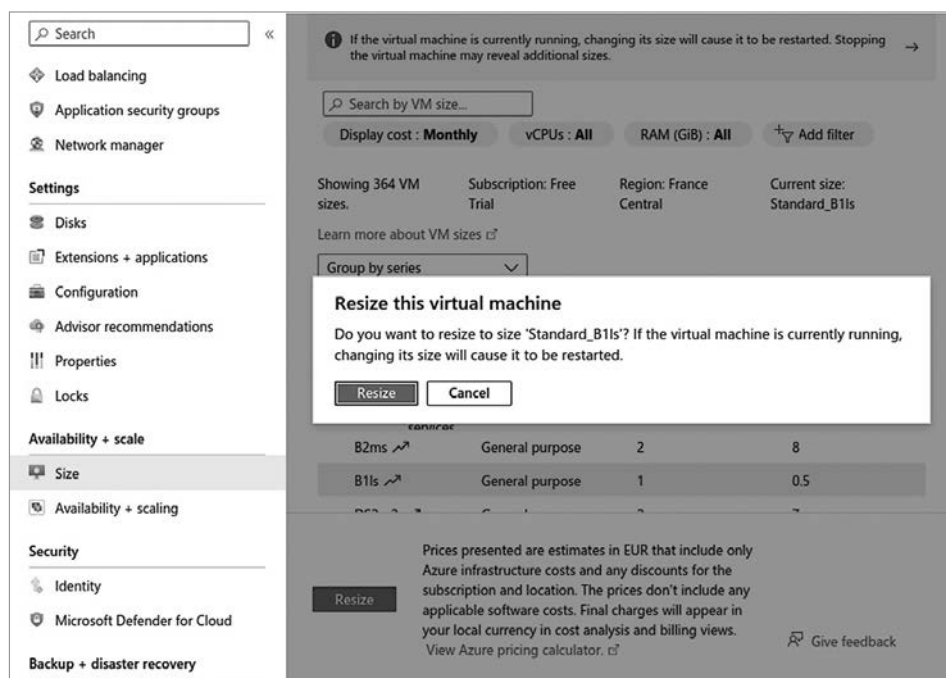


Рис. 15.32. Изменение размера виртуальной машины

Наша виртуальная машина (`packt-ubuntu-demo`) имеет размер **B1s** (1 виртуальный процессор, 1 Гбайт памяти), и этот размер можно увеличить или уменьшить.

В качестве упражнения мы изменим размер на меньший B1ls (1 процессор, 0,5 Гбайт памяти). Выберите вариант B1ls и нажмите **Resize** (Изменить размер). При этом Azure остановит виртуальную машину и перезапустит ее с новыми параметрами. Чем меньше размер экземпляра, тем он дешевле. Перед тем как изменять размер, всегда полезно останавливать машину, чтобы не нарушить целостность данных в экземпляре.

Одна из замечательных особенностей Azure состоит в том, что виртуальные машины можно масштабировать — в том числе наращивать емкость хранилища, добавляя дополнительные диски данных. Можно добавлять существующие диски или создавать новые.

Давайте разберемся, как добавить дополнительный диск к виртуальной машине.

Как добавить дополнительное хранилище

Azure позволяет добавлять диски в экземпляр на лету, не останавливая машину. Дополнительные сведения о типах дисков в Azure можно найти на странице docs.microsoft.com/azure/virtual-machines/disks-types.

Чтобы добавить диск данных, сделайте так:

1. В левой навигационной панели перейдите к разделу **Virtual machines** и выберите свой экземпляр. В разделе **Settings** (Настройки) щелкните **Disks** (Диски), а затем — **Create and attach a new disk** (Создать и присоединить новый диск).
2. В свойствах диска оставьте номер логической единицы (LUN) по умолчанию и задайте имя диска (например, `packt-disk`), тип хранилища и размер (например, 4 Гбайт). Когда закончите, нажмите **Apply** (Применить) (рис. 15.33).
3. После этого новый диск подключен к виртуальной машине, но на нем еще не инициализирована файловая система. Ее можно настроить и смонтировать так же, как рассказывалось в подразделе «Как настроить том хранилища EBS» выше в этой главе.

Теперь новый диск данных можно использовать как обычное файловое хранилище.

Обратите внимание, что диски данных сохраняются, только если виртуальная машина работает. Если ее приостановить, перевести в спящий режим или удалить, диск данных становится недоступным. Когда машина удаляется, диск теряется навсегда.

До сих пор мы выполняли все административные операции через веб-интерфейс на портале Azure. Но что, если вам нужно автоматизировать их с помощью сценариев? Для этого существует интерфейс командной строки Azure.

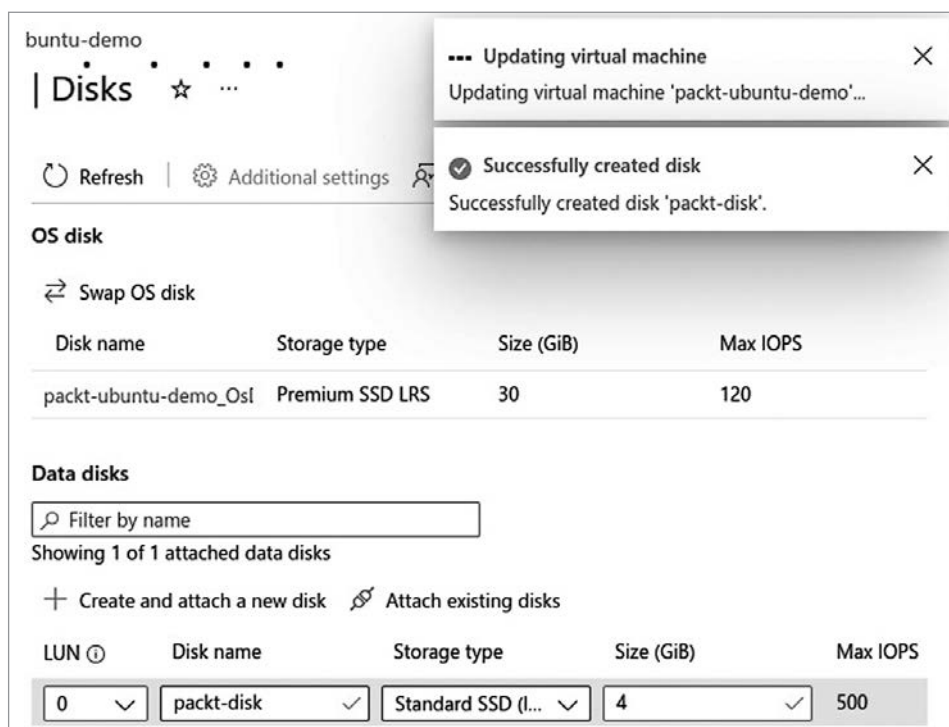


Рис. 15.33. Добавление нового диска данных

Интерфейс командной строки Azure CLI

С помощью интерфейса командной строки Azure (Azure CLI) можно управлять облачными ресурсами. Чтобы установить Azure CLI, следуйте инструкциям на странице learn.microsoft.com/cli/azure/install-azure-cli. В качестве упражнения мы установим его на компьютер под управлением Debian. Соответствующее пособие доступно по адресу learn.microsoft.com/cli/azure/install-azure-cli-linux. Azure CLI можно установить несколькими способами, из которых мы выберем такую команду:

```
curl -sL https://aka.ms/InstallAzureCLIDeb | sudo bash
```

Когда установка завершится, Azure CLI можно будет вызывать с помощью команды `az`. Например, такая команда выводит встроенную справку:

```
az help
```

Правда, прежде чем выполнять любые административные операции, нам нужно аутентифицировать CLI с помощью учетных данных Azure. Это можно сделать такой командой¹:

```
az login
```

Система предложит вам зайти на страницу microsoft.com/devicelogin и ввести определенный код, а также выбрать метод аутентификации. Если вы успешно аутентифицируетесь, то увидите сообщение наподобие такого:

```
Microsoft
Microsoft Azure Cross-platform Command Line Interface
You have signed in to the Microsoft Azure Cross-platform Command Line Interface
application on your device. You may now close this window.
```

Теперь мы готовы использовать Azure CLI для того, чтобы управлять облачными ресурсами. Давайте создадим новую группу ресурсов под названием `packt-dev` в регионе Центральная Франция:

```
alexandru@debian:~$ az group create --name packt-dev --location francecentral
{
  "id": "/subscriptions/d82a0c6e-...9d/resourceGroups/packt-dev",
  "location": "francecentral",
  "managedBy": null,
  "name": "packt-dev",
  "properties": {
    "provisioningState": "Succeeded"
  },
  "tags": null,
  "type": "Microsoft.Resources/resourceGroups"
}
```

Теперь запустим виртуальную машину Ubuntu с именем `packt-ubuntu-dev` в контексте этой группы ресурсов:

```
az vm create \
  --resource-group packt-dev \
  --name packt-ubuntu-dev \
  --image Ubuntu2204 \
  --admin-username packt \
  --generate-ssh-keys
```

Вот что означают ключи этой команды:

<code>resource-group</code>	Имя группы ресурсов, в которой создается виртуальная машина
<code>name</code>	Имя виртуальной машины

¹ Официальная документация по Azure не рекомендует аутентифицироваться с помощью `az login` и предлагает более безопасные методы: learn.microsoft.com/cli/azure/authenticate-azure-cli. — *Примеч. ред.*

image	Используемый образ дистрибутива Linux
admin-username	Имя пользователя учетной записи администратора машины
generate-ssh-keys	Сгенерировать новую пару ключей SSH для доступа к машине

Эта команда выводит сведения о новой машине в формате JSON. Когда создается виртуальная машина, Azure автоматически генерирует файлы ключей SSH и помещает их в каталог `~/.ssh` на локальной машине. В выводе также указан публичный IP-адрес машины в таком формате:

```
"publicIpAddress": "51.103.100.132"
```

Чтобы просмотреть список всех своих виртуальных машин, запустите такую команду:

```
az vm list
```

Если вы зайдете на портал Azure в браузере, то тоже увидите все свои виртуальные машины, включая только что созданную:

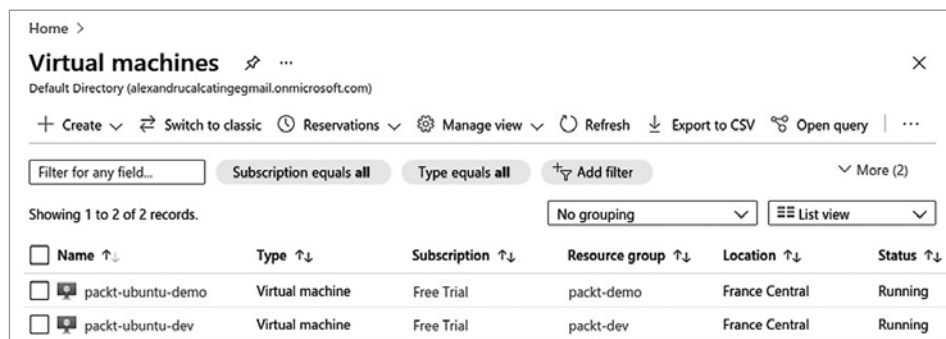


Рис. 15.34. Просмотр виртуальных машин на портале Azure

Чтобы получить информацию о конкретной виртуальной машине (например, `packt-ubuntu-dev`), запустите такую команду:

```
az vm show \
  --resource-group packt-dev \
  --name packt-ubuntu-dev
```

Чтобы заново развернуть существующую виртуальную машину, запустите такую команду:

```
az vm redeploy \
  --resource-group packt-dev \
  --name packt-ubuntu-dev
```

А эта команда удаляет виртуальную машину:

```
az vm delete \  
  --resource-group packt-dev \  
  --name packt-ubuntu-dev
```

Обратите внимание, что в командах, которые связаны с виртуальными машинами (`az vm`), необходимо указывать группу ресурсов, к которой принадлежит машина.

В этой главе мы изучили только основы интерфейса командной строки Azure. Чтобы получить более подробную информацию, обратитесь к официальной документации: learn.microsoft.com/cli/azure.

На этом мы завершаем тему развертывания в публичных облаках AWS и Azure. Мы затронули обширную область и обзорно рассмотрели операции, связанные с управлением облачными средами. Мы рекомендуем вам углубить свои знания и ознакомиться с дополнительными источниками, начиная с документации по облачным решениям AWS и Azure, ссылки на которые приведены в разделе «Дополнительная литература».

Итоги

AWS и Azure предоставляют примерно одинаковый набор функций и позволяют гибко настраивать вычислительные мощности, хранилища и сетевую инфраструктуру с оплатой по факту использования. Оба провайдера предлагают ключевые возможности публичного облака: эластичность, автоматическое масштабирование, оснащение в режиме самообслуживания, средства безопасности, управление идентификацией и доступом. Мы рассмотрели и ту и другую службу с практической точки зрения, сосредоточившись на типичных повседневных задачах развертывания и управления.

В этой главе вы узнали, как запускать и останавливать новые экземпляры или виртуальные машины, а также как регулировать вычислительную мощность экземпляра и масштабировать хранилища, присоединяя дополнительные блочные устройства (тома). Наконец, вы освоили инструменты командной строки, с помощью которых можно писать сценарии, чтобы автоматизировать различные задачи по управлению облаком.

К этому моменту вы должны быть знакомы с административными веб-консолями и интерфейсом командной строки AWS и Azure. Мы рассмотрели некоторые типичные задачи по управлению облаком и поговорили о том, как оснащать облачные ресурсы. В результате вы приобрели специализированные навыки современного администрирования Linux, которые помогут вам работать с облачными средами. Материал, который вы изучили, вместе со знаниями, полученными из предыдущих глав, формирует эффективный инструментарий для того, чтобы с помощью Linux управлять локальными, публичными и гибридными облачными системами.

В следующей главе мы продолжим эту тему и познакомимся с тем, как управлять развертыванием приложений с помощью контейнерных рабочих процессов и служб Kubernetes.

Вопросы

Давайте вспомним некоторые понятия, о которых вы узнали в этой главе:

1. Что такое зона доступности?
2. Какой тип экземпляров AWS EC2 обеспечивает лучшую производительность — `t3.small` или `t3.micro`?
3. Допустим, вы запустили экземпляр AWS EC2 в зоне доступности `us-west-1a` и собираетесь присоединить к нему том EBS, который создан в `us-west-1b`. Сработает ли это?
4. Как подключиться по SSH к экземпляру AWS EC2 или к виртуальной машине Azure?
5. Какая команда Azure CLI выводит список виртуальных машин? А как выглядит эквивалентная команда AWS CLI?
6. Какая команда AWS CLI позволяет запустить новый экземпляр EC2?
7. Какая команда Azure CLI позволяет удалить виртуальную машину?

Дополнительная литература

Вот несколько ресурсов, которые помогут вам глубже изучить облачные технологии AWS и Azure:

- Документация по AWS EC2: docs.aws.amazon.com/ec2.
- Документация по Azure: learn.microsoft.com/azure.
- Прашант Лакхера (Prashant Lakhera). «AWS for System Administrators».
- Ауробиндо Саркар (Aurobindo Sarkar), Амит Шах (Amit Shah). «Learning AWS, Second Edition».
- Джефф Уэббер-Кросс (Geoff Webber-Cross). «Learning Microsoft Azure».
- Виджай Саини (Vijay Saini). «Learning Microsoft Azure: A Hands-On Training», видеокурс.

16

Как развертывать приложения с помощью Kubernetes

Будь вы опытный системный администратор, который управляет контейнерными приложениями, или инженер DevOps, который автоматизирует рабочие процессы их оркестровки, **Kubernetes** может принести вам большую пользу. В этой главе вы познакомитесь с этой системой и научитесь создавать и настраивать **кластеры**. Мы покажем, как с помощью Kubernetes можно запускать и масштабировать простое приложение в безопасной и высокодоступной среде. Кроме того, вы узнаете, как взаимодействовать с Kubernetes с помощью интерфейса командной строки.

К концу этой главы вы научитесь устанавливать и настраивать кластеры Kubernetes, а также управлять ими на локальном компьютере. Кроме того, мы продемонстрируем, как развернуть и масштабировать приложение с помощью Kubernetes.

Вот обзор тем, которые мы рассмотрим в этой главе:

- Архитектура Kubernetes и объектная модель API.
- Как установить и настроить Kubernetes.
- Как работать с Kubernetes с помощью инструмента командной строки `kubectl` и развертывать приложения.

Технические требования

Вы должны быть знакомы с Linux и командной строкой в целом. Вам также будет значительно легче изучать Kubernetes, если вы уверенно ориентируетесь в **сетевых технологиях TCP/IP** и контейнерах **Docker**.

Понадобится следующее техническое оснащение:

- Локальный настольный компьютер с любым дистрибутивом Linux, чтобы устанавливать инструменты командной строки и упражняться с ними. В наших примерах будет использоваться Debian и Ubuntu LTS.

- Если в вашем распоряжении есть мощная настольная система (как минимум 8-ядерный процессор и 16 Гбайт оперативной памяти), вы сможете воспроизвести на ней необходимую среду; мы посвятим довольно большой раздел тому, как создать кластер Kubernetes на основе виртуальных машин.
- Гипервизор на настольном компьютере.

Итак, давайте начнем знакомство с Kubernetes.

Введение в Kubernetes

Kubernetes — это система с открытым исходным кодом для **оркестровки контейнерных приложений**. Первоначально его разработала компания Google. Оркестровка заключается в том, что система автоматически управляет контейнерными приложениями — в том числе оснащает, развертывает и масштабирует их. Если в приложении используются контейнерные микросервисы, система оркестровки предоставляет такие функции:

- **Эластичная оркестровка (автоматическое масштабирование)**. Службы приложений (контейнеры) автоматически запускаются и останавливаются в зависимости от конкретных требований и условий. Например, когда увеличивается количество запросов, могут запускаться дополнительные экземпляры веб-сервера, а когда количество запросов падает ниже определенного порога, ненужные серверы отключаются.
- **Управление рабочей нагрузкой**. Службы приложений оптимально развертываются и распределяются в пределах кластера, чтобы обеспечить обязательные зависимости и резервирование. Например, на каждом узле кластера может запускаться конечная точка веб-сервера, чтобы поддерживать высокую доступность.
- **Абстракция инфраструктуры** обеспечивает среду выполнения контейнера, сетевую инфраструктуру и балансировку нагрузки. Например, система может распределять нагрузку между несколькими контейнерами с веб-серверами и автоматически конфигурировать сетевое соединение с контейнером приложения базы данных.
- **Декларативная конфигурация** описывает и обеспечивает желаемое состояние многоуровневого приложения. Например, веб-сервер должен быть готов обслуживать запросы только тогда, когда его база данных находится в рабочем состоянии, а хранилище доступно.

Классический пример оркестровки рабочей нагрузки — служба потокового видео по запросу. Когда новое телешоу пользуется большим спросом, количество запросов может быть значительно выше среднего значения в течение сезона. Kubernetes позволяет масштабировать количество веб-серверов в зависимости от интенсивности потокового вещания. Также можно регулировать масштабирование некоторых компонентов промежуточного уровня, таких как экземпляры баз данных (которые обслуживают запросы на аутентификацию) и кэш-

хранилища (которые обслуживают потоки). Когда телешоу выходит из моды и количество запросов значительно снижается, Kubernetes автоматически отключает избыточные экземпляры, в результате чего сокращается зона развертывания приложения, а следовательно, снижаются издержки.

Вот некоторые ключевые преимущества развертывания приложений с помощью Kubernetes:

- **Быстрое развертывание.** Контейнеры приложений создаются и запускаются относительно быстро, используя либо **декларативную**, либо **императивную** модель конфигурации. (Мы поговорим об этом подробнее в разделе «Объектная модель Kubernetes» далее в этой главе.)
- **Быстрые итерации.** Приложения обновляются без особых хитростей: нижележащая инфраструктура просто плавно заменяет соответствующий контейнер.
- **Быстрое восстановление.** Если приложение выходит из строя или становится недоступным, Kubernetes автоматически восстанавливает его до нужного состояния, заменяя соответствующий контейнер.
- **Сокращение эксплуатационных расходов.** Контейнерная среда и инфраструктурная абстракция Kubernetes позволяют обходиться минимальными усилиями, чтобы администрировать и обслуживать систему, а также тратить относительно мало ресурсов, чтобы запускать приложения.

Теперь, когда вы получили общее представление о Kubernetes, давайте рассмотрим, как работает эта система.

Архитектура Kubernetes

Архитектурная модель Kubernetes опирается на три основных принципа:

- **Декларативная конфигурация** описывает желаемое состояние системы, то есть условия, которые необходимо соблюсти, чтобы приложение достигло рабочего режима: общее состояние приложения и микросервисов, развертывание необходимых контейнеров и сопутствующих ресурсов, включая сеть, хранилище и балансировщики нагрузки.
- **Контроллеры (управляющие циклы)** следят за тем, чтобы система находилась в желаемом состоянии, и при необходимости корректируют ее: например, заменяют контейнер приложения, который вышел из строя, или добавляют дополнительные ресурсы для возросших рабочих нагрузок.
- **Объектная модель API** представляет фактическую реализацию желаемого состояния, для чего использует различные конфигурационные объекты, которые взаимодействуют между собой с помощью API.

Чтобы лучше понять, как устроен Kubernetes, давайте подробнее рассмотрим его объектную модель и соответствующий API. Также обратите внимание на рис. 16.1, где наглядно изображена архитектура кластера Kubernetes.

Объектная модель Kubernetes

Архитектура Kubernetes определяет набор объектов, которые представляют желаемое состояние системы. Здесь **объект** — это программная сущность, которая описывает поведение подсистемы. Различные объекты взаимодействуют друг с другом через API, чтобы формировать желаемое состояние с течением времени. Другими словами, объектная модель Kubernetes — это программное представление желаемого состояния.

Итак, что же представляют собой эти объекты Kubernetes? К наиболее важным из них относятся сервер API, модули (поды), контроллеры, службы и хранилища. В последующих разделах мы расскажем обо всех этих объектах подробнее.

С помощью объектов можно настраивать состояние системы, используя декларативную или императивную модель конфигурации:

- В **декларативной модели** мы описываем состояние системы — обычно с помощью конфигурационного файла, или манифеста (в формате YAML или JSON). Такая конфигурация может охватывать и развертывать много объектов API, воспринимая систему как единое целое.
- В **императивной модели** мы вызываем отдельные команды для того, чтобы настраивать и развертывать конкретные объекты API, которые обычно действуют на один целевой объект или подсистему.

Давайте рассмотрим сервер API — центральный элемент объектной модели Kubernetes.

Что такое сервер API

Сервер API выступает в качестве конечной точки, которая позволяет управлять желаемым состоянием системы. Он предоставляет интерфейс HTTP REST, через который передаются данные в формате JSON. К серверу API можно обратиться в двух режимах:

- **изнутри** к нему обращаются другие объекты API;
- **извне** к нему обращаются рабочие процессы конфигурирования и управления.

Сервер API — это, по сути, шлюз, через который можно взаимодействовать с кластером Kubernetes как извне, так и изнутри. Кластер — это структура различных узлов, на которых выполняются контейнерные приложения. Это основная структура, которую обслуживает Kubernetes; мы расскажем об этом подробнее в разделе «Как устроен кластер Kubernetes». Системный администратор подключается к конечной точке сервера API, чтобы настраивать кластер и управлять им (как правило, с помощью командной строки). На внутреннем уровне объекты API Kubernetes подключаются к серверу API, чтобы сообщать об обновлениях своего состояния. В свою очередь, сервер может корректировать внутреннюю конфигурацию объектов API, чтобы привести ее к желаемому состоянию.

Объекты API — это основные компоненты, из которых состоит внутренняя конфигурация (желаемое состояние) кластера Kubernetes. Давайте рассмотрим несколько таких объектов.

Поды в Kubernetes

Под (pod) — это основная рабочая единица Kubernetes, которая функционирует как приложение, развернутое в одном или нескольких контейнерах. Поды также являются атомарными единицами планирования в Kubernetes. Другими словами, система обеспечивает, чтобы контейнеры из одного и того же пода совместно развертывались на одном узле кластера.

По сути, под — это микросервис (или служба) в сетке служб приложения. Например, в кластере типичного веб-приложения могут быть запущены такие поды:

- веб-сервер (nginx);
- аутентификация (Vault);
- база данных (PostgreSQL);
- хранилище данных (NAS).

Каждая из этих служб (или приложений) работает в своем поде. Несколько подов одного и того же приложения (например, веб-сервера) образуют коллекцию **ReplicaSet**, которую мы рассмотрим подробнее в разделе «Контроллеры Kubernetes».

Основные характеристики подов таковы:

- Они имеют **эфемерную природу**. Как только под завершает работу, он навсегда удаляется, и его нельзя развернуть снова. Таким образом, поды не сохраняют состояния, если только они не размещают свои данные в постоянном хранилище или на локальном томе.
- Поды — это **атомарные единицы**: они либо развернуты, либо нет. Для пода с одним контейнером атомарность практически сама собой разумеется. А если в нем несколько контейнеров, атомарность означает, что под развертывается только тогда, когда развернут каждый из них. Если какой-нибудь контейнер не удалось развернуть, под тоже не будет развернут, а значит, не будет и пода как такового. Если в действующем многоконтейнерном поде один контейнер выходит из строя, то весь под завершает работу.
- Чтобы следить за тем, как функционируют приложения внутри подов, Kubernetes использует **датчики**, или **пробы** (такие как датчик работоспособности или датчик готовности). Они нужны потому, что если под развернут и запущен, это не обязательно означает, что приложение или служба в нем работают нормально. Например, в поде веб-сервера может быть датчик, который передает запросы к определенному URL и в зависимости от ответа делает вывод, функционирует ли этот адрес.

Kubernetes отслеживает состояние подов с помощью контроллеров, о которых мы поговорим в следующем разделе.

Контроллеры Kubernetes

Контроллеры в Kubernetes — это управляющие циклы, которые постоянно наблюдают за состоянием кластера и обеспечивают, чтобы система поддерживалась в желаемом состоянии или приближалась к нему. Например, если контроллер обнаружит, что под не отвечает, он может развернуть новый под и утилизировать старый. Два основных типа контроллеров — **ReplicaSet** и **Deployment**:

- Контроллер может добавлять поды определенного типа в коллекцию реплик подов и удалять их из коллекции. Такие контроллеры называются **ReplicaSet**, и они отвечают за то, чтобы поддерживать определенное количество реплик подов в зависимости от текущего состояния приложения. Например, допустим, что приложению нужно три пода веб-сервера, и датчик показывает, что один из них стал недоступным. В этом случае контроллер ReplicaSet удаляет отказавший под и создает вместо него новый.
- Когда приложения развертываются в Kubernetes, поды обычно не создаются напрямую с помощью ReplicaSets. Вместо этого используется контроллер **Deployment**, который обеспечивает развертывания Kubernetes. В декларативной модели Kubernetes можно определить контроллер Deployment с одним или несколькими контроллерами ReplicaSet. Задача контроллера Deployment — создать ReplicaSet с необходимым количеством подов и управлять его состоянием. Другими словами, Deployment контролирует, какой образ контейнера загружать и сколько подов создавать.

Контроллер Deployment также помогает переходить от одного ReplicaSet к другому, прежде всего в сценариях развертывания или обновления. Представьте, что у нас есть ReplicaSet (v1) с несколькими подами, в которых работает версия 1 нашего приложения, и мы хотим обновить их до версии 2. Напоминаем, что поды нельзя пересоздавать или обновлять. Вместо этого мы определим второй ReplicaSet (v2) с подами версии 2. Контроллер Deployment утилизирует набор реплик v1 и иницирует v2. Kubernetes выполнит развертывание плавно, с минимальными перебоями в работе или вообще без них, а также может откатить обновление, если нужно.

В Kubernetes есть и другие типы контроллеров, и мы рекомендуем вам ознакомиться с ними на странице kubernetes.io/docs/concepts/workloads/controllers.

По мере того как приложения масштабируются или свертываются, соответствующие подсистемы развертываются или удаляются. Давайте рассмотрим службы Kubernetes, которые помогают управлять этими подсистемами.

Службы Kubernetes

Службы (Services) обеспечивают непрерывный доступ к приложениям, которые запущены в подах. Службы отвечают за то, чтобы поды были доступными: для этого службы маршрутизируют трафик к соответствующим конечным точкам приложений. Другими словами, службы предоставляют абстракцию сети для того, чтобы обмениваться данными с подами с помощью IP-адресов, маршру-

тизации и разрешения имен DNS. По мере того как поды развертываются или удаляются в зависимости от желаемого состояния системы, Kubernetes динамически обновляет конечную точку служб для подов, и при этом доступ к соответствующим приложениям практически не нарушается. Когда пользователи и приложения обращаются к постоянному IP-адресу конечной точки, служба обеспечивает, чтобы информация о маршрутизации была актуальной и трафик перенаправлялся только на исправные поды. С помощью службы также можно балансировать нагрузку между подами и масштабировать их (в большую или меньшую сторону) в зависимости от потребностей.

До сих пор мы рассматривали объекты API Kubernetes, которые контролируют развертывание приложений, доступ к ним и жизненный цикл служб. А как насчет постоянных данных, которые нужны приложениям? Давайте рассмотрим хранилище Kubernetes.

Хранилище Kubernetes

Kubernetes предоставляет различные типы хранилищ для приложений, которые работают в кластере. Самые распространенные типы — **тома** и **постоянные тома**. Поскольку поды эфемерны, данные приложения, которые хранятся в поде с помощью обычных томов, теряются, когда под завершает работу. А постоянные тома определяются и управляются на уровне кластера Kubernetes, и они не зависят от подов. Если приложениям (подам) требуется сохранять состояние, они резервируют постоянный том (определенного размера) с помощью механизма PVC (Persistent Volume Claim, запрос постоянного тома). Когда под, который использует постоянный том, завершает работу, новый под, развернутый вместо старого, получает текущее состояние из постоянного тома и продолжает использовать то же хранилище.

Подробнее о типах хранилищ Kubernetes можно узнать на странице kubernetes.io/docs/concepts/storage.

Теперь, когда мы познакомились с объектной моделью API Kubernetes, давайте обзорно рассмотрим архитектуру кластера Kubernetes.

Как устроен кластер Kubernetes

Кластер Kubernetes состоит из одного **управляющего узла** (Control Plane node) и одного или нескольких **рабочих узлов** (worker node). На следующей схеме представлена архитектура кластера в общем виде (рис. 16.1).

На этой схеме показаны два основных компонента кластера Kubernetes — рабочий и управляющий узлы. Эти узлы состоят из различных элементов: например, в рабочий узел входит контейнерная среда выполнения, кублет и объект kube-проху, а в управляющий узел — сервер API, система управления контроллерами и планировщик. Мы подробнее рассмотрим все эти элементы в следующих разделах.

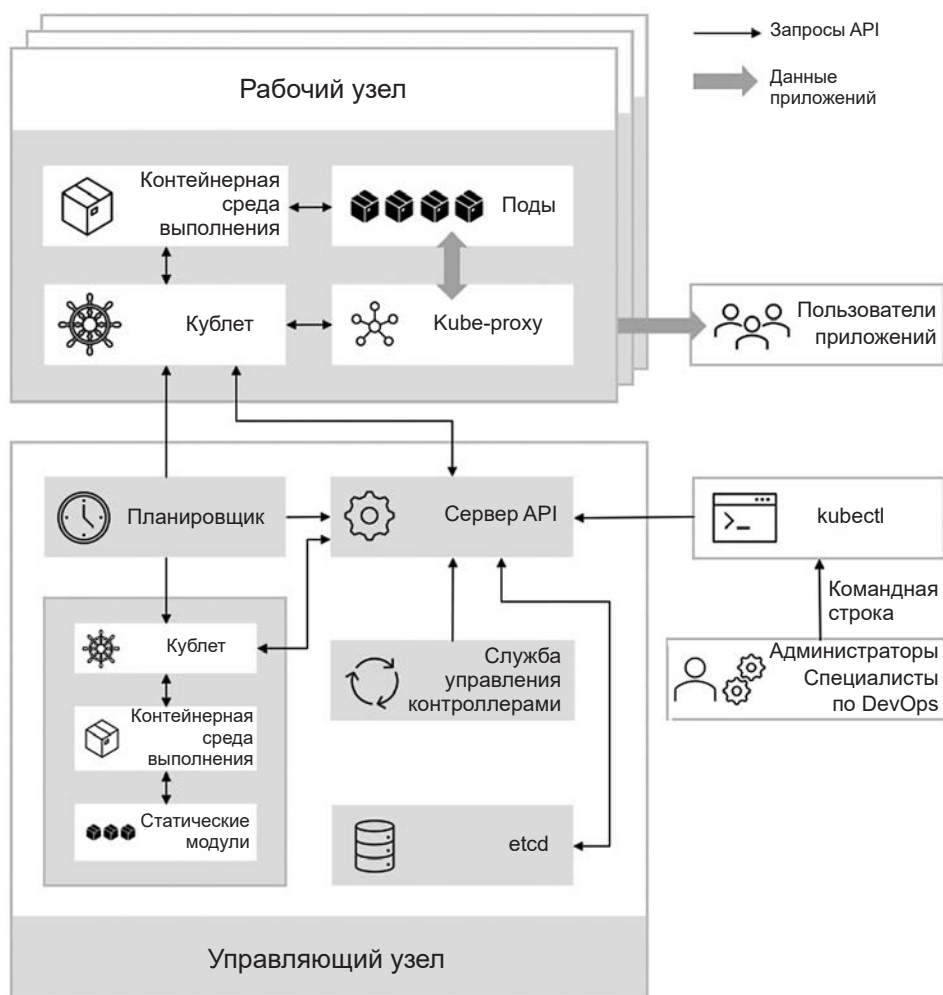


Рис. 16.1. Архитектура кластера Kubernetes

Для начала поговорим об узлах кластера Kubernetes, начиная с управляющего узла.

Управляющий узел Kubernetes

Управляющий уровень (Control Plane, CP) предоставляет основные службы, которые разворачивают и оркестрируют рабочие нагрузки приложений, и работает на выделенном узле кластера Kubernetes (управляющем узле), который реализует основные компоненты кластера, такие как планирование ресурсов и мониторинг. Этот узел также служит основной точкой доступа для администрирования кластера. Вот ключевые подсистемы управляющего узла:

- **Сервер API** — это центральный пункт связи между объектами API Kubernetes. Он также обеспечивает конечную точку для управления кластером, которая доступна либо через командную строку, либо через административную веб-консоль Kubernetes.
- **Планировщик** определяет, когда и на каких узлах развертывать поды, в зависимости от распределения ресурсов и административных политик.
- **Система управления контроллерами** поддерживает управляющие циклы, чтобы отслеживать и формировать желаемое состояние системы.
- **etcd** (кластерное хранилище) — это высокодоступная база данных в формате «ключ — значение», которая поддерживает состояние кластера Kubernetes и связанные с ним объекты API.
- **kubectl** — основной административный интерфейс командной строки, с помощью которого можно управлять кластером Kubernetes и взаимодействовать с ним. **kubectl** обменивается данными напрямую с сервером API, а также может удаленно подключаться к кластеру.

Детальный обзор архитектуры Kubernetes выходит за рамки этой главы. Все подробности описываются в официальной документации: kubernetes.io/docs/concepts/architecture.

Далее мы рассмотрим рабочий узел Kubernetes — основной исполнительный компонент кластера.

Рабочие узлы Kubernetes

Рабочие узлы кластера Kubernetes запускают фактические поды приложений и поддерживают их полный жизненный цикл. Эти узлы обеспечивают, чтобы поды получали вычислительную мощность Kubernetes, а рабочие нагрузки равномерно распределялись по кластеру по мере того, как развертываются и запускаются новые поды. Узлы можно сконфигурировать как физическое оборудование или как виртуальные машины.

Перечислим ключевые элементы узла Kubernetes:

- **Кублет** обрабатывает запросы управляющего узла (от планировщика) на развертывание и запуск подов приложения, а также следит за состоянием узлов и подов и сообщает серверу API о соответствующих изменениях.
- **Kube-Proxy** динамически настраивает виртуальную сетевую среду для приложений, которые выполняются в подах: маршрутизирует сетевой трафик, балансирует нагрузку и поддерживает IP-адреса служб и подов.
- **Контейнерная среда выполнения** обеспечивает среду, в которой выполняются поды в качестве контейнеров приложений. Она взаимодействует с нижележащим контейнерным движком (например, **containerd** или **Docker**) с помощью интерфейса CRI.

Все вышеперечисленные элементы работают на *каждом* узле кластера Kubernetes, в том числе на управляющем. На нем они необходимы специализированным подам, которые отвечают за управляющие службы, такие как DNS, балансировка нагрузки и веб-консоль.

Подробнее об узлах Kubernetes и соответствующих архитектурных понятиях можно узнать на странице kubernetes.io/docs/concepts/architecture/nodes.

Теперь, когда мы изучили некоторые ключевые понятия и компоненты кластера, давайте установим и настроим Kubernetes.

Как установить и настроить Kubernetes

Перед тем как устанавливать Kubernetes и работать с ним, нужно определиться, какую инфраструктуру вы будете использовать, — локальную или облачную. Кроме того, понадобится выбрать между моделями **IaaS** и **PaaS**. В первом случае вам придется самостоятельно устанавливать, настраивать и сопровождать кластер Kubernetes на физическом оборудовании или на виртуальных машинах. Это не самый простой вариант, и стоит тщательно продумать, как вы будете его реализовывать. А если вы выберете решение PaaS, которое доступно у всех крупных провайдеров публичных облаков, вам останутся только административные задачи и не придется обслуживать нижележащую инфраструктуру.

В этой главе мы рассмотрим развертывание Kubernetes по модели IaaS (инфраструктура как услуга). В упражнениях будем использовать настольную систему с виртуальными машинами Ubuntu.

При локальной установке также можно выбирать между облегченной настольной версией Kubernetes и полноценным кластером с несколькими узлами. Далее мы рассмотрим распространенные настольные версии Kubernetes.

Как установить Kubernetes на настольный компьютер

Если вы хотите только поэкспериментировать с Kubernetes, подойдет настольная версия, которая разворачивает кластер с одним узлом на локальной машине. Существует множество дистрибутивов Kubernetes для разных платформ, будь то Windows, macOS или Linux. Вот лишь некоторые из них:

Дистрибутив	ОС	Официальный сайт
Docker Desktop	Linux macOS Windows	docker.com/products/docker-desktop
minikube		minikube.sigs.k8s.io
MicroK8s		microk8s.io
K3s	Linux	k3s.io

В этом разделе мы продемонстрируем, как установить MicroK8s — один из самых популярных настольных дистрибутивов Kubernetes на момент написания этой книги. В качестве базовой операционной системы будем использовать Debian 12, поэтому сможем установить MicroK8s из Snap Store. Если у вас на машине нет `snapt`, установите соответствующий пакет с помощью `apt`:

```
sudo apt install snapd
```

Возможно, вам также понадобится установить среду выполнения ядра `snapt` с общими библиотеками и зависимостями:

```
sudo snap install core
```

После того как `snapt` установлен, можно установить MicroK8s с помощью такой команды:

```
alexandru@debian:~$ sudo snap install microk8s --classic
microk8s (1.32/stable) v1.32.3 from Canonical✓ installed
```

Чтобы получить доступ к командной строке MicroK8s без полномочий `sudo`, добавьте учетную запись локального пользователя в группу `microk8s`, а также измените права на каталог `~/ .kube`:

```
sudo usermod -aG microk8s $USER
sudo chown -fr $USER ~/ .kube
```

Изменения вступят в силу при следующем входе в систему, и вы сможете использовать утилиту командной строки `microk8s` без `sudo`. Например, такая команда отображает перечень подкоманд этой утилиты:

```
microk8s help
```

Чтобы увидеть состояние локального кластера MicroK8s с одним узлом, запустите такую команду:

```
microk8s status
```

В следующем разделе мы покажем, как установить MicroK8s на виртуальную машину. На этот раз в качестве базовой операционной системы в этой машине мы будем использовать Ubuntu 22.04 LTS.

Как создать кластер Kubernetes из виртуальных машин

В этом разделе мы приблизимся к реальной среде Kubernetes (хотя в довольно скромных масштабах) и развернем кластер Kubernetes на основе виртуальных машин под управлением Ubuntu. Для этого можно использовать любой гипервизор, например KVM, VirtualBox или VMware Fusion. В наших примерах будет фигурировать KVM.

Мы создадим четыре виртуальные машины и оснастим каждую из них 2 ядрами виртуального процессора, 2 Гбайт памяти и 20 Гбайт дискового пространства. Чтобы самостоятельно создать машины для своего гипервизора, обратитесь к разделу «Как установить Linux в виртуальной машине» главы 1 «Как установить Linux».

Прежде чем устанавливать кластер Kubernetes, давайте вкратце рассмотрим нашу учебную среду.

Учебная среда для кластера Kubernetes

Вот характеристики нашей среды VM:

Гипервизор	KVM
Кластер Kubernetes	1 управляющий узел 3 рабочих узла
Управляющий узел	k8s-cp1:192.168.122.104
Рабочие узлы	k8s-n1:192.168.122.146 k8s-n2:192.168.122.233 k8s-n3:192.168.122.163
Виртуальные машины	Ubuntu Server 22.04.3 LTS: 2 виртуальных процессора, 2 Гбайт памяти, 20 Гбайт дискового пространства
Пользователь (на каждом узле)	packt, с доступом по SSH

В каждой виртуальной машине мы задали имя пользователя и узла в мастере установки сервера Ubuntu. Также не забудьте включить сервер OpenSSH, когда появится соответствующий запрос. Скорее всего, IP-адреса ваших виртуальных машин будут отличаться от тех, которые указаны в нашей таблице. Кроме того, вы можете задать статические IP-адреса.

Чтобы упростить разрешение имен узлов в кластере, добавьте соответствующие записи в файл `/etc/hosts` на каждом узле. Например, на управляющем узле этот файл будет выглядеть так:

```
packt@k8s-cp1:~$ cat /etc/hosts
127.0.0.1      localhost
127.0.1.1      k8s-cp11
192.168.122.146 k8s-n1
192.168.122.233 k8s-n2
192.168.122.163 k8s-n3
```

¹ На каждом узле следует задать адрес `127.0.0.1` для самого этого узла и адреса в локальной сети для остальных узлов. — *Примеч. ред.*

В средах эксплуатации, где на узлах кластера работает межсетевой экран, нужно настроить его правила, которые разрешают входящий трафик по протоколу TCP на указанные порты (как описывается на странице kubernetes.io/docs/reference/networking/ports-and-protocols):

Порты	Назначение	Кто использует
Управляющий узел		
6443	Сервер API Kubernetes	Все
2379–2380	Клиентский API сервера etcd	kube-apiserver, etcd
10250	API кублетов	Текущий узел, управляющий узел
10259	kube-scheduler	Текущий узел
10257	kube-controller-manager	Текущий узел
Рабочие узлы		
10250	Kubelet API	Текущий узел, управляющий узел
10256	kube-proxy	Текущий узел, балансировщики нагрузки
30000–32767	Службы NodePort	Все

Рис. 16.2. Порты для узлов кластера Kubernetes

В следующих разделах предполагается, что виртуальные машины созданы и работают в соответствии с этими спецификациями. Прежде чем приступать к следующим шагам, стоит сделать несколько снапшотов виртуальных машин. Если в процессе установки что-то пойдет не так, вы сможете вернуть машину в исходное состояние и начать все сначала.

Вот что нужно сделать, чтобы установить кластер Kubernetes:

1. Отключить подкачку страниц.
2. Установить `containerd`.
3. Установить различные пакеты, которые относятся к Kubernetes, — например, `kubelet` и `kubeadm`.

Эти шаги нужно выполнить на каждом узле кластера¹. Начнем с того, что отключим подкачку страниц.

Как отключить подкачку страниц

Файл подкачки — это дисковое пространство, которое используется при переполнении памяти. Пакет `kubelet` на платформах Linux не работает, если вклю-

¹ Можно настроить все общие свойства на одной виртуальной машине, а затем клонировать ее в нужном количестве (см. раздел «Как клонировать виртуальные машины» в главе 11) и отрегулировать индивидуальные параметры каждой машины. — *Примеч. ред.*

цена подкачка (см. github.com/kubernetes/kubernetes/issues/53533), а значит, ее придется отключить. Для этого запустите на каждой виртуальной машине такую команду:

```
sudo swapon -a
```

Чтобы эта настройка сохранялась при перезагрузках системы, в файле `/etc/fstab` нужно закомментировать точки монтирования типа `swap`, например:

```
# /swap.img none swap sw 0 0
```

Для этого можно отредактировать файл вручную или использовать такую команду:

```
sudo sed -i '/\s*swap\s*/s/^(\.*)$/# \1/g' /etc/fstab
```

Не забудьте, что все это нужно сделать на каждом узле кластера. Давайте теперь посмотрим, как установить контейнерную среду выполнения Kubernetes.

Как установить *containerd*

`containerd` — это контейнерная среда выполнения по умолчанию в последних версиях Kubernetes. `containerd` реализует интерфейс CRI, который необходим для уровня абстракции контейнерного движка Kubernetes. Установить его не очень просто, и соответствующие процедуры могут меняться в зависимости от версий компонентов, которые в них задействованы, поэтому сверяйтесь с официальной документацией Kubernetes: <https://kubernetes.io/docs/setup/production-environment/container-runtimes/>. Среду выполнения нужно установить на каждом узле. Мы сначала установим необходимые компоненты на управляющем узле, потом на виртуальной машине `k8s-cp1`, а затем и на других узлах.

Давайте сначала выполним предварительную настройку для `containerd`:

1. Включите модули ядра `br_netfilter` и `overlay` с помощью `modprobe`:

```
sudo modprobe br_netfilter
sudo modprobe overlay
```

2. Сделайте так, чтобы эти модули загружались при перезагрузке системы:

```
cat <<EOF | sudo tee /etc/modules-load.d/containerd.conf
br_netfilter
overlay
EOF
```

3. Затем задайте параметры `sysctl`, которые нужны для CRI и тоже должны сохраняться при перезагрузке:

```
cat <<EOF | sudo tee /etc/sysctl.d/containerd.conf
net.bridge.bridge-nf-call-iptables = 1
net.bridge.bridge-nf-call-ip6tables = 1
net.ipv4.ip_forward = 1
EOF
```

4. Запустите следующую команду, чтобы все изменения вступили в силу немедленно, без перезагрузки:

```
sudo systemctl --system
```

5. Проверьте, включены ли нужные системные переменные в конфигурации `sysctl`. Если все в порядке, каждая из них должна иметь значение 1:

```
alexandru@debian:~$ sudo sysctl net.bridge.bridge-nf-call-iptables \
net.bridge.bridge-nf-call-ip6tables \
net.ipv4.ip_forward
```

```
net.bridge.bridge-nf-call-iptables = 1
net.bridge.bridge-nf-call-ip6tables = 1
net.ipv4.ip_forward = 1
```

6. Прежде чем устанавливать новые пакеты, обновите репозиторий `apt`:

```
sudo apt update
```

7. Теперь можно установить `containerd`:

```
sudo apt install -y containerd
```

8. Создайте конфигурацию `containerd` по умолчанию:

```
sudo mkdir -p /etc/containerd
containerd config default | sudo tee /etc/containerd/config.toml
```

Эти команды создадут новый каталог, а внутри него — новый конфигурационный файл `config.toml`. Вы увидите на экране достаточно объемный вывод, который покажет содержимое этого файла.

9. Нужно немного изменить стандартную конфигурацию `containerd`, чтобы использовать драйвер контрольных групп `systemd` с контейнерной средой выполнения (`runc`). Это необходимо потому, что базовая платформа (в нашем случае Ubuntu) использует подсистему инициализации `systemd`. Откройте файл `/etc/containerd/config.toml` в текстовом редакторе, например `nano`:

```
sudo nano /etc/containerd/config.toml
```

10. Найдите такой раздел (он находится внутри раздела `[plugins]`):

```
[plugins.'io.containerd.grpc.v1.cri'.containerd.runtimes.runc.options]
```

11. Добавьте в конец этого раздела следующую строку с нужным отступом (это *очень* важно):

```
[plugins.'io.containerd.grpc.v1.cri'.containerd.runtimes.runc.
options]
  SystemdCgroup = true
```

12. Сохраните файл `/etc/containerd/config.toml` и перезапустите `containerd`:

```
sudo systemctl restart containerd
```

13. Через несколько секунд можно проверить состояние службы `containerd` с помощью такой команды:

```
sudo systemctl status containerd
```

Если все в порядке, то вывод покажет, что система работает и проблем нет.

После того как `containerd` установлен и настроен, можно начать устанавливать пакеты Kubernetes.

Как установить пакеты Kubernetes

Чтобы установить пакеты Kubernetes, мы воспользуемся инструкцией с официального сайта: kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm. Напоминаем, что с момента выхода книги процедура могла измениться, поэтому сверяйтесь с современной документацией. Описанные далее шаги применимы к Debian 12 и Ubuntu 22.04:

1. Установите пакеты, которые нужны для репозитория Kubernetes:

```
sudo apt install -y apt-transport-https ca-certificates curl
```

2. Загрузите открытый ключ подписи **GPG** из репозитория. Укажите самую свежую версию Kubernetes (в нашем случае — 1.32):

```
sudo curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.32/deb/Release.key |  
sudo gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg
```

3. Добавьте репозиторий Kubernetes в систему:

```
echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg] https://  
pkgs.k8s.io/core:/stable:/v1.32/deb/ /' | sudo tee /etc/apt/sources.list.d/  
kubernetes.list
```

4. Давайте посмотрим, какие пакеты доступны в репозитории, который мы только что добавили:

```
sudo apt update -y
```

5. Теперь наконец можно установить пакеты Kubernetes:

```
sudo apt install -y kubelet kubeadm kubectl
```

6. Давайте зафиксируем версии этих пакетов, чтобы они не обновлялись через системные обновления безопасности и другие инструменты. Пакеты Kubernetes нужно обновлять исключительно с помощью процедур обновления кластера. Чтобы зафиксировать версии пакетов (в том числе `containerd`), используйте команду `apt-mark hold`:

```
packt@k8s-cpl:~$ sudo apt-mark hold containerd kubelet kubeadm kubectl  
containerd set on hold.  
kubelet set on hold.  
kubeadm set on hold.  
kubectl set on hold.
```

7. Наконец, настройте службы `containerd` и `kubelet` так, чтобы они запускались при загрузке системы:

```
sudo systemctl enable containerd
sudo systemctl enable kubelet
```

8. Теперь, когда пакеты Kubernetes установлены, проверьте состояние служб узла. Сначала опросим службу `containerd` и убедимся, что она активна и запущена:

```
packt@k8s-cpl:~$ sudo systemctl status containerd
● containerd.service - containerd container runtime
   Loaded: loaded (/usr/lib/systemd/system/containerd.service; enabled;
   preset: enabled)
   Active: active (running) since Wed 2025-04-09 18:13:16 UTC; 2min 30s ago
   Invocation: 029ca16663394b5fa73247f5e3943228
     Docs: https://containerd.io
   Main PID: 8828 (containerd)
     Tasks: 7
   Memory: 13.2M (peak: 16.3M)
      CPU: 112ms
   CGroup: /system.slice/containerd.service
           └─8828 /usr/bin/containerd
```

9. А теперь проверьте состояние службы `kubelet`:

```
sudo systemctl start kubelet && sudo systemctl status kubelet
```

Не удивляйтесь, если эта команда сообщит об ошибке: `kubelet` ищет кластер Kubernetes, но не может его найти, потому что он еще не настроен. Мы настроим кластер в следующем разделе с помощью `kubeadm`.

Прежде чем переходить к следующему разделу, убедитесь, что необходимые пакеты Kubernetes установлены на *всех* узлах кластера.

Далее мы загрузим (инициализируем) кластер Kubernetes с помощью `kubeadm`.

Как инициализировать кластер с помощью `kubeadm`

`kubeadm` — это вспомогательный инструмент, который позволяет управлять кластерами Kubernetes. Он выполняет две функции:

<code>kubeadm init</code>	Загрузить или инициализировать кластер
<code>kubeadm join</code>	Добавить узел в кластер

Команда `kubeadm init` по умолчанию выполняет такие задачи:

1. **Предварительные проверки.** На самом начальном этапе `kubeadm init` убеждается, что в системе есть минимально необходимые ресурсы процессора и памяти, настроены нужные права пользователей и есть поддерживаемая

контейнерная среда выполнения, которая совместима с CRI. Если любая из этих проверок не проходит, процедура создания кластера останавливается, а если все в порядке, то `kubeadm` переходит к следующему шагу.

2. **Создать удостоверяющий центр (CA).** `kubeadm init` создает самоподписанный удостоверяющий центр; с его помощью Kubernetes будет генерировать сертификаты, которые необходимы, чтобы аутентифицировать и запускать доверенные рабочие нагрузки в кластере. Файлы удостоверяющего центра хранятся в каталоге `/etc/kubernetes/pki/` и передаются на каждый узел, который присоединяется к кластеру.
3. **Сгенерировать конфигурационные файлы,** которыми пользуются компоненты кластера Kubernetes, чтобы обнаруживать сервер API, обмениваться данными с ним и аутентифицироваться. `kubeadm init` создает стандартный набор конфигурационных файлов, которые необходимы, чтобы загрузить кластер. Эти файлы хранятся в каталоге `/etc/kubernetes/`.
4. **Сгенерировать манифесты статических подов.** Статические поды — это специфические для системы поды, которые работают исключительно на управляющем узле под управлением демона `kubelet`, например сервер API, система управления контроллерами, планировщик и `etcd`. Манифесты статических подов — это конфигурационные файлы, которые описывают поды управляющего узла. `kubeadm init` генерирует манифесты статических подов во время загрузки кластера. Файлы манифестов хранятся в каталоге `/etc/kubernetes/manifests/`. Служба `kubelet` отслеживает это местоположение и, когда обнаруживает манифест, развертывает соответствующий статический под.
5. **Дождаться запуска статических подов.** После того как демон `kubelet` развернул статические поды, `kubeadm` запрашивает `kubelet` об их состоянии. Когда они запущены, `kubeadm init` переходит к следующему этапу.
6. **Пометить управляющий узел как отторгаемый.** Узлу присваивается специальный признак — ограничение (`taint`), из-за которого система не будет включать его в набор пользовательских подов. Противоположным понятием в Kubernetes является **толерантность**, которая обеспечивает, чтобы поды стремились привязываться к определенным узлам кластера. `kubeadm init` следует хорошему тону Kubernetes: пометить управляющий узел, чтобы на нем не запускались пользовательские поды. Очевидно, это важно для того, чтобы сохранить ресурсы управляющего узла исключительно для специфических системных рабочих нагрузок.
7. **Сгенерировать загрузочный токен.** Это простой токен, с помощью которого новые узлы присоединяются к кластеру Kubernetes. `kubeadm init` генерирует загрузочный токен, который можно передать доверенному узлу, чтобы он присоединился к кластеру.
8. **Запустить дополнительные поды.** Это специальные поды управляющего узла, которые расширяют функциональность кластера. По умолчанию `kubeadm init` создает и развертывает поды с дополнениями *DNS* и *kube-proxy*.

Процедуру загрузки кластера Kubernetes можно гибко настраивать. Если вызвать команду `kubeadm init` без дополнительных параметров, она запускает все задачи в описанном порядке. Однако системный администратор может вызывать ее с различными параметрами, чтобы тонко регулировать весь процесс.

Чтобы получить дополнительную информацию о `kubeadm`, обратитесь ко встроенной справке с помощью такой команды:

```
kubeadm help
```

Чтобы подробнее узнать о том, как загружать кластер Kubernetes с помощью `kubeadm`, в том числе устанавливать его, устранять неполадки и настраивать компоненты, можно обратиться к официальной документации по Kubernetes: kubernetes.io/docs/setup/production-environment/tools/kubeadm.

В следующем разделе мы развернем кластер Kubernetes в два этапа: сначала с помощью `kubeadm` мы создадим конфигурационный файл, а затем с помощью `kubeadm init` инициализируем эту конфигурацию. Давайте посмотрим, как создать управляющий узел Kubernetes.

Как создать управляющий узел Kubernetes

На управляющем узле мы задействуем решение **Calico**, которое обеспечит сетевое окружение для контейнеров и подов. Затем мы сгенерируем кластер по умолчанию с помощью `kubeadm`, а после этого будем использовать различные инструменты, чтобы применить конфигурацию. Мы выбрали Calico чисто субъективно, как одно из средств управления коммуникациями между рабочими нагрузками и компонентами. Чтобы подробнее узнать об этом продукте, посетите страницу docs.tigera.io/calico/latest/about.

Команды в этом упражнении выполняются на виртуальной машине `k8s-cp1`, которую мы выбрали в качестве управляющего узла нашего кластера Kubernetes. Теперь давайте настроим этот узел:

1. Сначала загрузите манифест Calico для **программно-определяемой сети (SDN)**: это логический сетевой уровень, который обеспечивает безопасное и прозрачное сетевое взаимодействие между подами через физическую сеть, которая может быть недоступна для настройки. Внутреннее устройство кластерной сети выходит за рамки этой главы, но мы рекомендуем прочесть о нем на странице kubernetes.io/docs/concepts/cluster-administration/networking. Чтобы загрузить манифест, запустите такую команду¹:

```
curl https://raw.githubusercontent.com/projectcalico/calico/v3.29.3/manifests/calico.yaml -O
```

¹ Здесь использована последняя версия Calico на момент подготовки этого издания. Узнать об обновлениях Calico можно в репозитории GitHub: github.com/projectcalico/calico. — *Примеч. ред.*

Эта команда загружает в текущий каталог (в нашем случае `/home/packt/`) файл `calico.yaml`, который пригодится нам позже, когда мы будем настраивать сетевое окружение модулей с помощью `kubect1`.

- Откройте файл `calico.yaml` в текстовом редакторе и обратите внимание на такие строки:

```
# - name: CALICO_IPV4POOL_CIDR
#   value: "192.168.0.0/16"
```

`CALICO_IPV4POOL_CIDR` — это пул бесклассовой адресации, который задает сетевой диапазон для модулей. Если указанная подсеть как-либо конфликтует с вашим локальным окружением, вам понадобится изменить ее здесь. Мы оставим эту настройку как есть.

- Создайте конфигурационный файл кластера по умолчанию с помощью `kubeadm`. Этот файл описывает настройки кластера Kubernetes, который мы собираем. Чтобы сгенерировать файл и присвоить ему имя `k8s-config.yaml`, запустите такую команду:

```
kubeadm config print init-defaults | tee k8s-config.yaml
```

- Внесите несколько изменений в файл `k8s-config.yaml`, который вы только что сгенерировали. Откройте его в текстовом редакторе и найдите параметр `localAPIEndpoint.advertiseAddress` — IP-адрес конечной точки сервера API. Значение по умолчанию `1.2.3.4` нужно заменить на IP-адрес виртуальной машины, на которой работает управляющий узел (`k8s-cp1`), — в нашем случае `192.168.122.104`. (См. раздел «Учебная среда для кластера Kubernetes» в начале этой главы.)

```
kind: InitConfiguration
localAPIEndpoint:
#   advertiseAddress: 1.2.3.4
   advertiseAddress: 192.168.122.104
bindPort: 6443
```

- Следующие правки понадобятся в разделе `nodeRegistration`, который идет сразу за `localAPIEndpoint`. Измените значение `criSocket` на сокет `containerd` (`/run/containerd/containerd.sock`) и укажите имя виртуальной машины (`k8s-cp1`):

```
nodeRegistration:
  criSocket: /run/containerd/containerd.sock
  imagePullPolicy: IfNotPresent
  name: k8s-cp1
  taints: null
```

- Далее изменим параметр `kubernetesVersion` в соответствии с версией нашей среды Kubernetes (ее можно узнать с помощью команды `kubeadm version`):

```
imageRepository: registry.k8s.io
kind: ClusterConfiguration
kubernetesVersion: 1.32.2
```

7. Наконец, присвойте параметру `cgroupDriver` (драйвер контрольной группы) для `kubelet` значение `systemd`, аналогичное драйверу для `containerd`. Обратите внимание, что системой управления службами в Ubuntu является `systemd`, поэтому управление соответствующей службой нужно передать демонам Kubernetes. В `k8s-config.yaml` пока нет соответствующего конфигурационного блока, так что добавьте его в конец файла вручную или с помощью такой команды:

```
cat <<EOF | cat >> k8s-config.yaml
---
apiVersion: kubelet.config.k8s.io/v1beta1
kind: KubeletConfiguration
cgroupDriver: systemd
EOF
```

8. Теперь все готово для того, чтобы загрузить кластер Kubernetes. Запустите команду `kubeadm init` с ключом `--config`, который указывает на файл конфигурации кластера (`k8s-config.yaml`):

```
sudo kubeadm init --config=k8s-config.yaml
```

Эта команда может выполняться несколько минут. Если кластер успешно загрузится, вы увидите такой вывод:

```
Your Kubernetes control-plane has initialized successfully!
# Ваш управляющий уровень Kubernetes успешно инициализирован
```

To start using your cluster, you need to run the following as a regular user:

```
# Чтобы начать использовать ваш кластер,
# запустите эти команды с правами обычного пользователя
```

```
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

You should now deploy a Pod network to the cluster.

```
# Теперь разверните сеть модулей на кластере
```

Run "`kubectl apply -f [podnetwork].yaml`" with one of the options listed at: </docs/concepts/cluster-administration/addons/>

You can now join any number of machines by running the following on each node as root:

```
# Вы можете присоединить любое количество машин,
# запустив следующую команду на каждом узле от имени root
```

```
kubeadm join 192.168.122.104:6443 \
--token 7bb4c3.25fe05a25378e218 \
--discovery-token-ca-cert-hash sha256:eb35fbe063ec0ec62883c9da
78d4e2607663ea24876af536cdeab19b6b6b4f65
```

Итак, наш управляющий узел Kubernetes запущен и работает. В этом выводе вы видите сообщение об удачном запуске, а также команды для следующих операций:

- задание текущего пользователя в качестве администратора кластера Kubernetes;
- подключение новых узлов к кластеру.

На примере этого упражнения мы рекомендуем вам уделить время тому, чтобы повторить последовательность задач `kubeadm init`, о которой рассказывалось в разделе «Как инициализировать кластер с помощью `kubeadm`» ранее в этой главе.

9. Чтобы задать текущего пользователя в качестве администратора кластера Kubernetes, запустите такие команды:

```
mkdir -p ~/.kube
sudo cp -i /etc/kubernetes/admin.conf ~/.kube/config
sudo chown $(id -u):$(id -g) ~/.kube/config
```

10. Теперь, когда кластер запущен, давайте развернем сетевой манифест Calico, чтобы создать сеть подов:

```
kubectl apply -f calico.yaml
```

Эта команда создает набор ресурсов, которые связаны с программно-определяемой сетью модулей.

11. После этого мы готовы впервые взглянуть на состояние нашего кластера и просмотреть все модули в системе с помощью `kubectl`:

```
packt@k8s-cp1:~$ kubectl get pods --all-namespaces
```

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE
kube-system	calico-kube-controllers-...	1/1	Running	0	3m16s
kube-system	calico-node-gwcn4	1/1	Running	0	3m16s
kube-system	coredns-5dd5756b68-28pgh	1/1	Running	0	4m35s
kube-system	coredns-5dd5756b68-j92f9	1/1	Running	0	4m35s
kube-system	etcd-k8s-cp1	1/1	Running	0	4m40s
kube-system	kube-apiserver-k8s-cp1	1/1	Running	0	4m40s
kube-system	kube-controller-manager-...	1/1	Running	0	4m40s
kube-system	kube-proxy-7bz9m	1/1	Running	0	4m35s
kube-system	kube-scheduler-k8s-cp1	1/1	Running	0	4m40s

Ключ `--all-namespaces` извлекает поды из всех групп ресурсов в кластере. Kubernetes организует ресурсы с помощью **пространств имен**. Пока в нашем кластере работают только **системные** поды, потому что мы еще не развернули ни одного **пользовательского** пода.

12. Следующая команда выводит текущие узлы в кластере:

```
packt@k8s-cp1:~$ kubectl get nodes
```

NAME	STATUS	ROLES	AGE	VERSION
k8s-cp1	Ready	control-plane	6m23s	v1.32.2

Вывод показывает, что в кластере Kubernetes настроен единственный узел `k8s-cp1`, который работает как управляющий.

13. Возможно, вы помните, что до того, как мы загрузили кластер, служба `kubelet` постоянно аварийно завершала работу (и пыталась перезапуститься). Теперь демон `kubelet` должен быть активен и запущен:

```
packt@k8s-cp1:~$ sudo systemctl status kubelet
● kubelet.service - kubelet: The Kubernetes Node Agent
   Loaded: loaded (/lib/systemd/system/kubelet.service; enabled; vendor
  preset: enabled)
   Drop-In: /usr/lib/systemd/system/kubelet.service.d-10
            └─kubeadm.conf
   Active: active (running) since Tue 2023-09-26 09:20:50 UTC; 6min ago
     Docs: https://kubernetes.io/docs/
   Main PID: 51436 (kubelet)
    Tasks: 10 (limit: 2220)
   Memory: 51.0M
      CPU: 5.399s
   CGroup: /system.slice/kubelet.service
           └─51436 /usr/bin/kubelet -- bootstrap-kubeconfig=/etc/
kubernetes/bootstrap-kubelet.conf
```

14. Просмотрите список манифестов, которые созданы в каталоге `/etc/kubernetes/manifests/` для каждого компонента кластера:

```
packt@k8s-cp1:~$ ls /etc/kubernetes/manifests/
etcd.yaml kube-apiserver.yaml kube-controller-manager.yaml
kube-scheduler.yaml
```

Здесь перечислены конфигурационные файлы, которые описывают статические (системные) модули — сервер API, систему управления контроллерами, планировщик и etcd.

15. Обратите внимание также на конфигурационные файлы в каталоге `/etc/kubernetes/`:

```
ls /etc/kubernetes/
```

Как вы помните из раздела «Как инициализировать кластер с помощью kubeadm» этой главы, эти файлы используются компонентами кластера, чтобы обмениваться данными с сервером API и аутентифицироваться на нем.

В этом разделе мы активно использовали утилиту `kubect1`. Ознакомьтесь с официальной документацией, чтобы подробнее узнать о ее подкомандах и ключах: kubernetes.io/docs/reference/generated/kubect1/kubect1-commands.

Далее давайте добавим в наш кластер рабочие узлы.

Как присоединить узел к кластеру Kubernetes

Прежде чем добавлять узел в кластер Kubernetes, необходимо выполнить предварительные действия, которые описывались в разделе «Учебная среда для кластера Kubernetes» ранее в этой главе.

Чтобы присоединить узел к кластеру, нам понадобятся как **загрузочный маркер**, так и **маркер обнаружения** (хеш-сумма сертификата удостоверяющего центра) — они оба были сгенерированы, когда мы инициализировали кластер. В листинге на с. 580 показан пример команды `kubeadm join` с обоими маркерами. Имейте в виду, что загрузочный маркер действует 24 часа. Если вы не скопировали

маркеры из вывода `kubeadm init`, их можно запросить командой `kubeadm token list`, как показано ниже.

Итак, чтобы присоединить узел к кластеру Kubernetes, выполните следующие действия в терминале управляющего узла (`k8s-cp1`):

1. Получите текущие загрузочные маркеры:

```
packt@k8s-cp1:~$ kubeadm token list
TOKEN                                TTL    EXPIRES                                USAGES
  DESCRIPTION  EXTRA GROUPS
7bb4c3.25fe05a25378e218    23h    2023-09-27T09:20:50Z    authentication,
signing
  <none>                system: bootstrappers:kubeadm:default-node-token
```

2. Получите хеш-сумму сертификата удостоверяющего центра:

```
packt@k8s-cp1:~$ openssl x509 -pubkey \
> -in /etc/kubernetes/pki/ca.crt | \
> openssl rsa -pubin -outform der 2>/dev/null | \
> openssl dgst -sha256 -hex | sed 's/^.* //'
eb35fbe063ec0ec62883c9da78d4e2607663ea24876af536cdeab19b6b6b4f65
```

3. Кроме того, можно сгенерировать новый загрузочный маркер с помощью такой команды:

```
kubeadm token create
```

Ключ `--print-join-command` позволяет распечатать полную команду `kubeadm join` с необходимыми параметрами:

```
kubeadm token create --print-join-command
```

Теперь, когда маркеры готовы, можно перейти к следующим шагам, на которых мы будем использовать наши начальные маркеры из листинга на с. 580.

1. Переключитесь в терминал рабочего узла (`k8s-n1`) и запустите такую команду. Обратите внимание, что без `sudo` она не работает:

```
sudo kubeadm join 192.168.122.104:6443 \
  --token 7bb4c3.25fe05a25378e218 \
  --discovery-token-ca-cert-hash sha256:af8e659bc8a9069cf58d3100e94644
bb6ee804c1372a0ab7e74801cf8340a15d
```

2. Чтобы проверить состояние текущих узлов в кластере, запустите такую команду в терминале управляющего узла (`k8s-cp1`):

```
packt@k8s-cp1:~$ kubectl get nodes
NAME      STATUS   ROLES    AGE   VERSION
k8s-cp1   Ready    control-plane   58m   v1.32.2
k8s-n1    Ready    <none>         2m45s v1.32.2
```

Как видите, к кластеру присоединился новый узел `k8s-n1`.

3. Присоедините аналогичным образом два других узла кластера (`k8s-n2` и `k8s-n3`). Если во время присоединения вы слишком быстро опрашиваете кластер с управляющего узла, вы можете временно увидеть состояние

`NotReady` для новых узлов, пока на них разворачиваются модули управляющего уровня. В конце концов в выводе команды `kubectl get nodes` (на `k8s-cp1`) все три узла должны отображаться в состоянии `Ready`.

Вот мы и установили кластер Kubernetes с одним управляющим и тремя рабочими узлами. Мы использовали виртуальные машины в локальной среде, но эти же процедуры применимы и к решениям IaaS, которые размещены в частном или публичном облаке.

В следующем разделе рассмотрим утилиту командной строки `kubectl` и узнаем, как с ее помощью создавать ресурсы Kubernetes и управлять ими. Затем поговорим о том, как разворачивать и масштабировать приложения в императивном и декларативном режиме разворачивания в Kubernetes.

Как работать с Kubernetes

В этом разделе мы рассмотрим реальные примеры взаимодействия с кластером Kubernetes. Поскольку мы будем интенсивно применять утилиту командной строки `kubectl`, мы подробно обсудим некоторые наиболее распространенные схемы ее использования. Затем перейдем к тому, как разворачивать приложения в кластере Kubernetes. Мы будем работать в локальной среде, которую настроили в разделе «Как создать кластер Kubernetes из виртуальных машин».

Для начала давайте подробнее рассмотрим `kubectl`.

Утилита командной строки `kubectl`

`kubectl` — это основной инструмент, с помощью которого управляют кластером Kubernetes и его ресурсами. `kubectl` взаимодействует с оконечной точкой сервера API, который работает в кластере, с помощью Kubernetes REST API. Общий синтаксис команды `kubectl` выглядит так:

```
kubectl [команда] [ТИП] [ИМЯ] [флаги]
```

В общем случае команды `kubectl` выполняют операции семейства **CRUD**¹ над ресурсами Kubernetes — такими как поды, разворачивания и службы.

`kubectl` выводит результаты в форматах YAML, JSON или обычного текста. Эти форматы удобны, чтобы создавать или редактировать манифесты разворачивания приложений. Можно записать вывод команды `kubectl` (например, той, которая создает ресурс) в формате YAML в файл, а затем повторно использовать этот файл, чтобы выполнить ту же операцию (или последовательность операций) в декларативном режиме. Это подводит нас к двум основным парадигмам разворачивания Kubernetes:

¹ Аббр. Create (создать), Read (читать), Update (обновить), Delete (удалить). — Примеч. пер.

- **Императивные развертывания.** Вызываются одна или несколько команд `kubectl`, которые работают с определенными ресурсами.
- **Декларативные развертывания.** Файлы манифеста развертываются с помощью команды `kubectl apply`, обычно для того, чтобы обработать набор ресурсов одним вызовом.

Мы подробнее рассмотрим эти две модели в разделе «Как развертывать приложения в Kubernetes» далее в этой главе. А пока давайте вернемся к команде `kubectl`. Вот ее распространенные подкоманды:

create	Создать ресурс (императивно)
apply	Создать ресурс (декларативно)
get	Перечислить ресурсы
edit	Обновить ресурс
set	Изменить конфигурацию ресурсов
delete	Удалить ресурсы
run	Запустить под
exec	Выполнить команду в поде
describe	Вывести подробную информацию о ресурсах
explain	Вывести документацию по ресурсам
logs	Вывести журнал контейнера в поде

Также упомянем часто используемые ключи `kubectl`:

--dry-run	Запускает команду в холостом режиме: она не изменяет состояние системы, но выводит результаты так, как если бы выполнялась нормально
--output	Формат вывода: <code>yaml</code> , <code>json</code> и <code>wide</code> (обычный текст с дополнительными сведениями)

В следующих разделах мы рассмотрим немало примеров использования команды `kubectl`. Она запускается по такой общей схеме:

	операция	ресурс		параметры
kubectl	[команда]	[ТИП]	[ИМЯ]	[флаги]
kubectl	get	pods	packt-web	--output=wide
kubectl	create	service	packt-svc	--dry-run

Рис. 16.3. Общая схема использования `kubectl`

С полным справочником по командам `kubectl` можно ознакомиться на странице kubernetes.io/docs/reference/kubectl. Вам также может пригодиться шпаргалка по наиболее распространенным командам и флагам `kubectl`: kubernetes.io/docs/reference/kubectl/quick-reference.

Теперь давайте подготовим среду `kubectl`, чтобы взаимодействовать с кластером Kubernetes, который мы создали ранее с виртуальными машинами. Если вы предпочитаете использовать `kubectl` на управляющем узле, можете пропустить следующий раздел.

Как подключиться к кластеру Kubernetes с локального компьютера

В этом разделе мы настроим и запустим утилиту командной строки `kubectl` в настольной системе Linux, чтобы управлять удаленным кластером Kubernetes. Наш компьютер работает под управлением Debian 12.

Чтобы установить `kubectl`, мы будем следовать инструкциям на странице kubernetes.io/docs/tasks/tools/install-kubectl-linux. Загрузите последнюю версию `kubectl` с помощью такой команды:

```
curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
```

Затем установите `kubectl`:

```
sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl
```

Теперь, когда пакет установлен, можно проверить его версию:

```
alexandru@debian:~$ kubectl version --client
Client Version: v1.32.3
Kustomize Version: v5.5.0
```

Теперь давайте добавим еще одну конфигурацию кластера в нашу среду. На этот раз мы подключимся к локальному управляющему узлу Kubernetes и обновим конфигурационные файлы с помощью `kubectl`. Вот как это сделать:

1. Сначала скопируйте конфигурационный файл с удаленного управляющего узла (в нашем случае `k8s-cp1, 192.168.122.104`) во временное расположение (допустим, `/tmp/config.cp`):

```
scp packt@192.168.122.104:~/.kube/config /tmp/config.cp
```

2. Затем переместите этот файл в нужное место:

```
mv /tmp/config.cp ~/.kube/config
```

3. Если хотите, можно удалить временные файлы:

```
rm ~/.kube/config.old /tmp/config.cp
```

4. Просмотрите текущие контексты конфигурации Kubernetes:


```
alexandru@debian:~$ kubectl config get-contexts
CURRENT  NAME                                CLUSTER    AUTHINFO    NAMESPACE
*        kubernetes-admin@kubernetes        kubernetes kubernetes-admin
```

Эта команда выводит сведения о нашем новом кластере Kubernetes с именем администратора (`kubernetes-admin@kubernetes`) и именем кластера (`kubernetes`).

- Для единообразия давайте изменим имя контекста локального кластера на `k8s-local` и сделаем его контекстом по умолчанию в нашей среде `kubectl`:

```
kubectl config rename-context \
    kubernetes-admin@kubernetes \
    k8s-local
kubectl config use-context k8s-local
```

Далее рассмотрим некоторые распространенные команды `kubectl`, которые используются в повседневных задачах администрирования Kubernetes.

Как работать с `kubectl`

Вот одна из первых команд, которую обычно запускают после того, как подключаются к кластеру Kubernetes:

```
kubectl cluster-info
```

Эта команда выводит IP-адрес и порт сервера API, который прослушивает управляющий узел, а также другую информацию:

```
alexandru@debian:~$ kubectl cluster-info
Kubernetes control plane is running at https://192.168.122.104:6443
CoreDNS is running at https://192.168.122.104:6443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy
```

Подкоманда `dump` выводит текущее состояние кластера, что помогает диагностировать и отлаживать связанные с ним проблемы:

```
kubectl cluster-info dump
```

Чтобы получить подробные сведения об узлах кластера, запустите команду `kubectl get nodes` с флагом `--output=wide` (или `-o wide`):

```
alexandru@debian:~$ kubectl get nodes --output=wide
NAME      STATUS ROLES    AGE  VERSION  INTERNAL-IP  EXTERNAL-IP  OS-IMAGE
KERNEL-VERSION  CONTAINER-RUNTIME
k8s-cp1 Ready  control-plane  6h44m v1.28.2  192.168.122.104 <none>      Ubuntu
22.04.3 LTS 5.15.0-84-generic containerd://1.7.2
k8s-n1 Ready  <none>         5h49m v1.28.2  192.168.122.146 <none>      Ubuntu
22.04.3 LTS 5.15.0-84-generic containerd://1.7.2
k8s-n2 Ready  <none>         5h43m v1.28.2  192.168.122.233 <none>      Ubuntu
22.04.3 LTS 5.15.0-84-generic containerd://1.7.2
k8s-n3 Ready  <none>         5h42m v1.28.2  192.168.122.163 <none>      Ubuntu
22.04.3 LTS 5.15.0-84-generic containerd://1.7.2
```

Следующая команда перечисляет поды, которые запущены в пространстве имен по умолчанию:

```
kubectl get pods
```

Сейчас у нас не запущен ни один пользовательский под, поэтому команда выводит такое сообщение:

```
No resources found in default namespace1.
```

Чтобы перечислить все поды, а не только пользовательские, к предыдущей команде можно добавить флаг `--all-namespaces`:

```
alexandru@debian:~$ kubectl get pods --all-namespaces
```

NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE
kube-system	calico-kube-controllers-...	1/1	Running	0	6h46m
kube-system	calico-node-4wfl4	1/1	Running	1 (5h51m ago)	5h52m
kube-system	calico-node-7769m	1/1	Running	0	5h46m
kube-system	calico-node-gwcn4	1/1	Running	0	5h46m
kube-system	calico-node-npkpr	1/1	Running	0	6h46m
kube-system	coredns-5dd5756b68-j92f9	1/1	Running	0	6h47m
kube-system	etcd-k8s-cp1	1/1	Running	0	6h47m
kube-system	kube-apiserver-k8s-cp1	1/1	Running	0	6h48m
kube-system	kube-controller-manager-k8s-cp1	1/1	Running	0	6h48m
kube-system	kube-proxy-7bz9m	1/1	Running	0	6h47m
kube-system	kube-proxy-c48p4	1/1	Running	0	5h46m
kube-system	kube-proxy-lmtkl	1/1	Running	0	5h46m
kube-system	kube-proxy-zmwfm	1/1	Running	1 (5h51m ago)	5h52m
kube-system	kube-scheduler-k8s-cp1	1/1	Running	0	6h48m

Эта команда выводит все поды, которые запущены в системе. Поскольку это исключительно системные поды, они связаны с пространством имен `kube-system`. Мы получили бы тот же результат, если бы указали флаг `--namespace kube-system`:

```
kubectl get pods --namespace kube-system
```

Чтобы получить полное представление обо всех ресурсах, которые работают в системе, запустите такую команду:

```
kubectl get all --all-namespaces
```

До сих пор мы упоминали только самые распространенные типы объектов, такие как узлы (`nodes`), поды (`pods`) и службы (`services`). Полный список всех типов можно просмотреть с помощью такой команды (здесь показана только часть вывода):

```
alexandru@debian:~$ kubectl api-resources
```

NAME	SHORTNAMES	APIVERSION	NAMESPACED	KIND
bindings		v1	true	Binding

¹ Не обнаружены поды в пространстве имен по умолчанию. — *Примеч. пер.*

componentstatuses	cs	v1	false	ComponentStatus
configmaps	cm	v1	true	ConfigMap
endpoints	ep	v1	true	Endpoints
events	ev	v1	true	Event
limitranges	limits	v1	true	LimitRange
namespaces	ns	v1	false	Namespace
nodes	no	v1	false	Node
persistentvolumeclaims	pvc	v1	true	
PersistentVolumeClaim				
persistentvolumes	pv	v1	false	PersistentVolume
Pods	po	v1	true	Pod
...				

В список входят названия типов объектов API (например, `nodes`), их краткие имена (например, `no`), а также сведения о том, можно ли организовать их в пространства имен (например, `false`).

Если вы хотите подробнее узнать о конкретных объектах API (допустим, об узлах), вам пригодится команда `explain`:

```
alexandru@debian:~$ kubectl explain nodes
```

```
KIND:      Node
```

```
VERSION:   v1
```

DESCRIPTION:

Node is a worker node in Kubernetes. Each node will have a unique identifier in the cache (i.e. in etcd).

FIELDS:

```
apiVersion    <string>
```

APIVersion defines the versioned schema of this representation of an object. Servers should convert recognized schemas to the latest internal value, and may reject unrecognized values. More info:

<https://k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#resources>

Эта команда выводит подробную документацию о типе объекта `nodes`, включая связанные с ним поля API. В предыдущем листинге показано поле `apiVersion`, которое описывает версиюрованную схему текущего представления объекта. Чтобы просмотреть документацию по отдельному полю, передайте это поле команде `explain`, например:

```
kubectl explain nodes.apiVersion
```

Привыкайте использовать `explain`, чтобы уточнять сведения о различных типах объектов API в кластере Kubernetes. Обратите внимание, что эта команда предоставляет документацию о *типах ресурсов*. Не путайте ее с командой `describe`, которая выводит подробную информацию о *самых ресурсах* в системе, например:

<code>kubectl describe nodes</code>	Информация обо всех узлах
<code>kubectl describe nodes k8s-n1</code>	Информация об узле k8s-n1

Каждую команду `kubectl` можно вызвать с ключом `--help` (или `-h`), чтобы получить контекстную справку, например:

```
kubectl --help
kubectl config -h
kubectl get pods -h
```

Утилита командной строки `kubectl` поддерживает довольно много команд, и может понадобиться некоторое время, чтобы ее освоить. Порой вам бывает нужно найти определенную команду или вспомнить, как правильно пишется ее имя или как ее использовать. Здесь поможет автозавершение Bash для `kubectl`, о котором мы поговорим в следующем подразделе.

Как включить автозавершение kubectl

Автозавершение `kubectl` выводит контекстно-зависимые предложения, если при вводе команды дважды нажать клавишу `Tab`.

Эта функция зависит от пакета `bash-completion`, который включен по умолчанию в большинстве платформ Linux. Если у вас его нет, установите его вручную; например, в Ubuntu это можно сделать так:

```
sudo apt-get install -y bash-completion
```

Затем подключите сценарий автозаполнения `kubectl` в профиле вашей командной оболочки:

```
echo "source <(kubectl completion bash)" >> ~/.bashrc
```

Изменения вступят в силу при следующем входе в терминал или сразу же, если обновить исходный профиль `bash`:

```
source ~/.bashrc
```

После того как вы настроили автозавершение `kubectl`, вы сможете получать контекстно-зависимые предложения, если дважды нажмете клавишу `Tab` при вводе команды. Например, так можно просмотреть все типы ресурсов, которые создаются с помощью команды `create`:

```
alexandru@debian:~$ kubectl create # ← Здесь вы дважды нажимаете Tab
clusterrole          (Create a cluster role)
clusterrolebinding   (Create a cluster role binding for a particular cluster
                      role)
configmap             (Create a config map from a local file, directory
                      or literal value)
cronjob              (Create a cron job with the specified name)
deployment            (Create a deployment with the specified name)
ingress              (Create an ingress with the specified name)
job                  (Create a job with the specified name)
namespace             (Create a namespace with the specified name)
poddisruptionbudget   (Create a pod disruption budget with the specified name)
priorityclass         (Create a priority class with the specified name)
```

quota	(Create a quota with the specified name)
role	(Create a role with single rule)
rolebinding	(Create a role binding for a particular role or cluster role)
secret	(Create a secret using a specified subcommand)
service	(Create a service using a specified subcommand)
serviceaccount	(Create a service account with the specified name)
token	(Request a service account token)

Автозавершение `kubectl` охватывает все синтаксические компоненты: команды, ресурсы (типы и имена) и флаги (ключи).

Теперь, когда мы знакомы с командой `kubectl`, пора перейти к тому, как разворачивать приложения в Kubernetes.

Как разворачивать приложения в Kubernetes

Когда в начале раздела «Утилита командной строки `kubectl`» мы представили команду `kubectl` и схему ее использования, мы упомянули о том, что ресурсы приложений в Kubernetes можно создавать двумя способами: императивным и декларативным.

В этом разделе мы подробно рассмотрим обе эти модели на примере разворачивания простого веб-приложения. Начнем с императивного режима.

Императивные разворачивания

Напомним, что при императивном разворачивании мы запускаем последовательность команд `kubectl`, которые создают необходимые ресурсы и приводят кластер в желаемое состояние (например, чтобы запустить приложение). Декларативные разворачивания приводят к тому же результату, но обычно с помощью одной команды `kubectl apply`, которой передается файл манифеста, описывающий несколько ресурсов.

Как создать разворачивание

Итак, давайте создадим разворачивание (объект `Deployment`). Мы назовем его `packt`, а в качестве приложения возьмем демонстрационный контейнер `nginx` из публичного реестра Docker (docker.io/nginxdemos/hello):

```
kubectl create deployment packt --image=nginxdemos/hello
```

Команда выводит сообщение о том, что разворачивание успешно создано:

```
deployment.apps/packt created
```

У нас получилось разворачивание с одним набором реплик (объект `ReplicaSet`), в котором содержится один под, а в нем запущено приложение веб-сервера. Отметим, что подом управляет система управления контроллерами в стеке разворачивания приложений (`deployment.apps`). Вместо этого можно было развер-

нуть простой под приложения (назовем его `packt-web`) с помощью такой команды:

```
kubectl run packt-web --image=nginxdemos/hello
```

В этом случае вывод покажет, что наш под `pod/packt-web` является автономным, то есть не входит в объект `Deployment`:

```
pod/packt-web created
```

Далее в этом разделе мы увидим, что этот под также не входит в `ReplicaSet` и поэтому не управляется системой управления контроллерами.

Давайте посмотрим на состояние нашей системы, запросив подробную информацию о подах:

```
alexandru@debian:~$ kubectl get pods -o wide
NAME                                READY STATUS  RESTARTS  AGE  IP              NODE    NOMINATED NODE  READINESS GATES
packt-579bb9c999-rtvzr             1/1   Running    0         58s  172.16.215.65  k8s-n1  <none>          <none>
packt-web                          1/1   Running    0         27s  172.16.57.193  k8s-n3  <none>          <none>
```

Этот вывод показывает, что наши поды запущены и Kubernetes развернул их на отдельных узлах кластера, чтобы сбалансировать нагрузку и оптимально распределить ресурсы в кластере:

Под	Узел
packt-579bb9c999-rtvzr	k8s-n1
packt-web	k8s-n3

`packt-579bb9c999-rtvzr` — это прикладной под, которым управляет контроллер. Kubernetes сгенерировал для него уникальное имя, добавив к имени разворачивания (`packt`) **хеш-сумму шаблона** пода (`579bb9c999`) и **идентификатор** пода (`rtvzr`). Оба этих значения уникальны в пределах набора реплик.

В отличие от этого, имя автономного пода (`packt-web`) не изменилось, потому что он не входит в разворачивание. Давайте вызовем команду `describe` для обоих подов, чтобы получить о них больше информации. Начнем с управляемого пода (не забудьте воспользоваться автозавершением `kubectl`, дважды нажав `Tab`):

```
alexandru@debian:~$ kubectl describe pod packt-579bb9c999-rtvzr
Name:                packt-579bb9c999-rtvzr
Namespace:           default
Priority:              0
Service Account:     default
Node:                 k8s-n1/192.168.122.146
Start Time:           Wed, 27 Sep 2023 15:00:48 +0300
Labels:               app=packt
                     pod-template-hash=579bb9c999
Annotations:          cni.projectcalico.org/containerID:
                     df487cdf4881c686e817406c4ab...
```

```

cni.projectcalico.org/podIP: 172.16.215.65/32
cni.projectcalico.org/podIPs: 172.16.215.65/32
Status:      Running
IP:          172.16.215.65
IPs:
  IP:        172.16.215.65
Controlled By: ReplicaSet/packt-579bb9c999
Containers:
  hello:
    Container ID: containerd://5da6f0c90504595e9c67d8fb107e7f39c4dc15f7eed4a1e8bf...
    Image:        nginxdemos/hello
    Image ID:     docker.io/nginxdemos/hello@sha256:dedfbe85183df66f3fdc99accf53e...
    Port:        <none>
    Host Port:    <none>
    State:       Running
      Started:   Wed, 27 Sep 2023 15:00:54 +0300
    Ready:       True

```

Та же команда для автономного пода (**packt-web**) выводит немного другой результат, в котором нет поля **Controlled By**:

```

alexandru@debian:~$ kubectl describe pod packt-web
Name:          packt-web
Namespace:     default
Priority:       0
Service Account: default
Node:          k8s-n3/192.168.122.163
Start Time:    Wed, 27 Sep 2023 15:01:19 +0300
Labels:        run=packt-web
Annotations:    cni.projectcalico.org/containerID: df5750e969ea7c707f6fe49797b9...
                cni.projectcalico.org/podIP: 172.16.57.193/32
                cni.projectcalico.org/podIPs: 172.16.57.193/32
Status:        Running
IP:            172.16.57.193
IPs:
  IP:          172.16.57.193
Containers:
  packt-web:
    Container ID: containerd://5da6f0c90504595e9c67d8fb107e7f39c4dc15f7eed4a1e8bfa...
    Image:        nginxdemos/hello
    Image ID:     docker.io/nginxdemos/hello@sha256:dedfbe85183df66f3fdc99accf53e1...
    Port:        <none>
    Host Port:    <none>
    State:       Running
      Started:   Wed, 27 Sep 2023 15:01:25 +0300
    Ready:       True
...

```

Можно также подключиться к любому узлу кластера, на котором запущены наши поды, и поближе рассмотреть связанные с ними контейнеры. Например, возьмем узел **k8s-n3 (192.168.122.163)**, на котором запущен автономный под (**packt-web**). Сначала войдите в терминал узла по SSH:

```
ssh packt@192.168.122.163
```

Затем с помощью среды выполнения `containerd` опросите контейнеры в системе:

```
alexandru@debian:~$ sudo crictl --runtime-endpoint unix:///run/containerd/
containerd.sock ps
CONTAINER          IMAGE                CREATED             STATE              NAME
1b1645d08ba49     2a4dceb283aa0       27 minutes ago     Running            packt-web
25e488ac7681c     8065b798a4d67       2 hours ago        Running            calico-node
6da991c5c8619     c120fed2beb84       2 hours ago        Running            kube-proxy
```

Далее мы рассмотрим, как получить доступ к процессам, которые запущены внутри подов.

Доступ к процессам в подах

Давайте переключимся обратно в локальную среду `kubectl` (на локальном компьютере, а не в виртуальной машине) и выполним следующую команду, чтобы получить доступ к оболочке контейнера, в котором запущен под `packt-web`:

```
kubectl exec -it packt-web -- /bin/sh
```

Эта команда переключает нас в интерактивную оболочку внутри контейнера, в которой можно запускать команды, как если бы мы подключились к узлу `packt-web` с помощью терминала. Интерактивный сеанс создается с помощью ключа `-it` (или `--interactive --tty`).

Давайте запустим несколько команд, начиная с обзора процессов:

```
/ # ps aux
PID USER      TIME COMMAND
  1 root        0:00 nginx: master process nginx -g daemon off;
 20 nginx     0:00 nginx: worker process
 21 nginx     0:00 nginx: worker process
 22 root        0:00 /bin/sh
 28 root        0:00 ps aux
```

Чтобы получить IP-адрес модуля, запустите такую команду:

```
/ # ifconfig | grep 'inet addr:' | cut -d: -f2 | awk '{print $1}' | \
    grep -v '127.0.0.1'
172.16.57.193
```

Имя узла можно получить с помощью команды `hostname`:

```
/ # hostname
packt-web
```

Чтобы выйти из оболочки контейнера, запустите команду `exit` или нажмите `Ctrl+D`. С помощью команды `kubectl exec` можно запустить внутри пода любой процесс, который в нем доступен.

Теперь давайте протестируем под приложения `packt-web` с помощью `curl`. Обратите внимание, что к конечной точке веб-сервера `packt-web` сейчас можно обратиться только по его внутреннему IP-адресу. Чтобы узнать этот адрес и другие

сведения о поде, мы использовали команды `kubectl get pods -o wide` и `describe`, но IP-адрес пода можно узнать и с помощью такой однострочной команды:

```
alexandru@debian:~$ kubectl get pods packt-web -o jsonpath='{.status.podIP}'{"\n"}'
172.16.57.193
```

В нашем случае команда выводит значение `172.16.57.193`. Мы использовали ключ `-o jsonpath`, чтобы передать запрос JSON для конкретного поля — `{.status.podIP}`. Не забывайте, что IP-адрес пода доступен только в его сети (`172.16.0.0/16`) внутри кластера.

Таким образом, чтобы обратиться к конечной точке `packt-web` с помощью команды `curl`, эта команда должна исходить из сети пода. Этого проще всего добиться, если запустить тестовый под с установленной утилитой `curl`:

1. Эта команда запускает под, который называется `test` и основан на образе `Docker curlimages/curl`:

```
alexandru@debian:~$ kubectl run test --image=curlimages/curl sleep 600
pod/test created
```

2. Мы искусственно поддерживаем жизнь контейнера с помощью команды `sleep`, потому что точка входа соответствующего образа `Docker` устроена так, что под просто выполняет команду `curl` и завершает работу. `sleep` позволяет отложить выполнение точки входа `curl`, чтобы под оставался доступен¹.
3. Теперь в поде `test` можно запустить простую команду `curl`, которая обращается к конечной точке веб-сервера `packt-web`:

```
alexandru@debian:~$ kubectl exec test -- curl http://172.16.57.193
<!DOCTYPE html>
<html>
<head>
<title>Hello World</title>
```

Мы получили ответ HTTP и соответствующие записи журнала доступа от сервера `nginx`, который запущен в поде.

4. Чтобы просмотреть журналы пода `packt-web`, запустите такую команду:

```
alexandru@debian:~$ kubectl logs packt-web
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to
perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
...
2023/09/27 12:01:25 [notice] 1#1: using the "epoll" event method
2023/09/27 12:01:25 [notice] 1#1: nginx/1.25.2
...
172.16.111.193 - - [27/Sep/2023:12:43:21 +0000] "GET / HTTP/1.1" 200 7231 "-"
"curl/8.3.0" "-"
```

¹ Чтобы под сохранялся неограниченно долго (например, когда это нужно для тестирования или отладки), часто используют команду `sleep infinity`. — *Примеч. ред.*

5. Журналы в поде `packt-web` создаются веб-сервером `nginx` и перенаправляются в стандартные потоки вывода (`stdout`) и ошибок (`stderr`) с помощью символических ссылок. Это легко проверить:

```
alexandru@debian:~$ kubectl exec packt-web -- ls -la /var/log/nginx
total 8
drwxr-xr-x    2 root root    4096 Aug 29 00:37 .
drwxr-xr-x    1 root root    4096 Aug 29 00:37 ..
lrwxrwxrwx    1 root root     11 Aug 29 00:37 access.log -> /dev/stdout
lrwxrwxrwx    1 root root     11 Aug 29 00:37 error.log -> /dev/stderr
```

6. Когда вы закончите экспериментировать с подом `test`, его можно удалить с помощью такой команды:

```
kubectl delete pods test
```

Теперь, когда мы развернули наше первое приложение в кластере Kubernetes, давайте посмотрим, как обеспечить доступ к соответствующей конечной точке из внешнего мира. Для этого можно открыть развертывание с помощью службы.

Как предоставить доступ к развертыванию с помощью службы

Вспомните команду, которую мы использовали, чтобы создать развертывание `packt` (не запускайте ее; она приведена здесь для справки):

```
kubectl create deployment packt --image=nginxdemos/hello
```

Эта команда привела к такому результату:

1. Было создано развертывание (объект `Deployment` с именем `packt`).
2. Объект развертывания создал набор реплик (объект `ReplicaSet` с именем `packt-579bb9c999`).
3. Набор реплик создал под (`packt-579bb9c999-rtvzr`).

Все это можно проверить с помощью таких команд:

```
kubectl get deployments -l app=packt
kubectl get replicaset -l app=packt
kubectl get pods -l app=packt
```

В этих командах мы использовали ключ `-l` (`--label-columns`), чтобы отфильтровать результаты по метке `app=packt`, которая обозначает ресурсы типа `Deployment` приложения `packt`.

Попробуйте подробнее изучить каждый из этих ресурсов с помощью команды `kubectl describe`. Не забывайте использовать автозавершение `kubectl` при вводе команд:

```
kubectl describe deployment packt | more
kubectl describe replicaset packt | more
kubectl describe pod packt-5dc77bb9bf-bnzsc | more
```

Команда `kubectl describe` может принести много пользы, когда вы устраняете неполадки в приложениях или развертываниях. Чтобы понять, почему поды не запускаются, выдают ошибки и ведут себя не так, как ожидается, просмотрите раздел **Events** («События») в выводе этой команды.

Теперь, когда мы развернули свое первое приложение в кластере Kubernetes, давайте посмотрим, как предоставить доступ к его конечной точке из внешнего мира.

Наше приложение `packt` состоит из одного пода `packt-579bb9c999-rtvzr`, а в нем запущен веб-сервер `nginx`, который прослушивает порт `80`. Как упоминалось ранее, пока доступ к поду можно получить только в пределах его сети, которая существует только внутри кластера. В этом разделе мы покажем, как открыть доступ к приложению (или развертыванию) для внешнего мира. Для этого в API Kubernetes предназначен объект службы (**Service**), который маршрутизирует сетевой трафик к подам приложения в развертывании. Вот как его можно создать и использовать:

1. Следующая команда создает службу для нашего развертывания:

```
alexandru@debian:~$ kubectl expose deployment packt
# Предоставить доступ к развертыванию packt
--port=80 \
# Связать порт 80 текущей службы...
--target-port=80 \
# ...с портом 80 внутри контейнера
--type=NodePort
# Сделать службу доступной вне кластера

# Сообщение о том, что служба успешно создана:
service/packt exposed
```

Без флага `--type=NodePort` типом службы по умолчанию станет `ClusterIP`, и ее конечная точка будет доступна только в пределах кластера.

2. Давайте рассмотрим нашу службу `packt` поближе:

```
alexandru@debian:~$ kubectl get service packt
NAME      TYPE        CLUSTER-IP      EXTERNAL-IP  PORT(S)          AGE
packt     NodePort    10.105.111.243  <none>       80:32664/TCP     28s
```

Эта команда выводит IP-адрес кластера, который назначен службе (`10.105.111.243`), и порты, которые она прослушивает для трафика TCP (`80:32664/TCP`):

80	Внутри кластера
32664	На любом узле вне кластера

Следует отметить, что IP-адрес кластера доступен только внутри кластера, а не извне.

Также не путайте внешний IP-адрес (**EXTERNAL-IP**) с IP-адресом узла кластера, по которому доступна наша служба. Внешний IP-адрес (который настраивается с помощью флага `--external-ip`) обычно соответствует балансировщику нагрузки, и его конфигурирует облачный провайдер, у которого размещен кластер Kubernetes.

3. Теперь наше приложение должно быть доступно извне кластера, и к нему можно обратиться из браузера, запросив любой узел кластера на порте 32664. Чтобы получить список узлов с IP-адресами и именами, можно запустить такую команду:

```
alexandru@debian:~$ kubectl get nodes \
-o jsonpath='{range .items[*]}{.status.addresses[*].address}{"\n"}'

192.168.122.104 k8s-cp1
192.168.122.146 k8s-n1
192.168.122.233 k8s-n2
192.168.122.163 k8s-n3
```

4. Давайте обратимся к управляющему узлу (`k8s-cp1`); для этого нужно ввести в браузере его адрес: `http://192.168.122.104:32664`.

Запрос HTTP из браузера передается конечной точке службы `packt`, которая маршрутизирует нужные сетевые пакеты к поду приложения (`packt-579bb9c999-rtvzr`). Веб-приложение `packt` отправляет ответ в виде стандартной веб-страницы **Hello World**, на которой отображается внутренний IP-адрес (`172.16.215.65`) и имя пода (`packt-579bb9c999-rtvzr`):



Рис. 16.4. Доступ к службе приложения `packt`

5. Чтобы убедиться, что веб-страница отражает верные сведения, можно запросить ту же информацию с помощью такой команды `kubectl`:

```
alexandru@debian:~$ kubectl get pod packt-579bb9c999-rtvzr \
-o jsonpath='{.status.podIP}{"\n"}{.metadata.name}{"\n"}'

172.16.215.65
packt-579bb9c999-rtvzr
```

Как видите, команда выводит тот же самый IP-адрес и имя пода.

Теперь предположим, что наше приложение принимает интенсивный трафик и мы хотим масштабировать набор реплик, который управляет подами. В следующем разделе мы покажем, как решить эту задачу.

Как масштабировать развертывания приложений

Сейчас в развертывании `packt` всего один под. Чтобы масштабировать развертывания приложения, нужно получить данные о запущенных репликах, масштабировать их до нужного количества и протестировать. Вот как это сделать:

1. Чтобы узнать, сколько реплик запущено, используйте такую команду:

```
alexandru@debian:~$ kubectl describe deployment packt
Name:                packt
Namespace:           default
CreationTimestamp:    Wed, 27 Sep 2023 15:00:48 +0300
Labels:              app=packt
Annotations:         deployment.kubernetes.io/revision: 1
Selector:            app=packt
Replicas:            1 desired | 1 updated | 1 total | 1 available | 0
unavailable
StrategyType:        RollingUpdate
MinReadySeconds:     0
RollingUpdateStrategy: 25% max unavailable, 25% max surge
```

2. Давайте увеличим развертывание `packt` до 10 реплик с помощью такой команды:

```
alexandru@debian:~$ kubectl scale deployment packt --replicas=10
deployment.apps/packt scaled
```

3. Если теперь вывести список подов в `packt`, мы увидим 10 запущенных подов:

```
alexandru@debian:~$ kubectl get pods -l app=packt
NAME                                READY   STATUS    RESTARTS   AGE
packt-579bb9c999-5zwnn              1/1     Running   0           22s
packt-579bb9c999-dnpm6              1/1     Running   0           22s
packt-579bb9c999-fw2fn              1/1     Running   0           22s
packt-579bb9c999-g5wwp              1/1     Running   0           22s
packt-579bb9c999-lrf74              1/1     Running   0           22s
packt-579bb9c999-r4npz              1/1     Running   0           22s
packt-579bb9c999-rfv5j              1/1     Running   0           22s
packt-579bb9c999-rtvzr              1/1     Running   0           174m
packt-579bb9c999-s6mr1              1/1     Running   0           22s
packt-579bb9c999-zdvl1              1/1     Running   0           22s
```

4. Входящие запросы к конечной точке службы нашего приложения (`http://192.168.122.104:32664`) будут распределяться по нагрузке между подами. Чтобы это пронаблюдать, можно использовать `curl` или текстовый браузер с интерфейсом командной строки. Обычный браузер не подойдет, потому что он, скорее всего, будет оптимизировать трафик и кэшировать ответы. В этом упражнении мы воспользуемся простым текстовым браузером

Lynx. В нашей настольной системе Debian 12 он уже установлен, а вы можете установить его с помощью такой команды:

```
sudo apt-get install -y lynx
```

5. Запросите в Lynx конечную точку нашего приложения:

```
lynx 172.16.191.6:32081
```

Если обновлять страницу с помощью **Ctrl + R** каждые несколько секунд, вы увидите, что адрес и имя сервера меняются в зависимости от того, какой под обрабатывает запрос:



Рис. 16.5. Балансировка нагрузки между подами

Чтобы выйти из Lynx, нажмите **Q**, а затем **Enter**.

6. Чтобы уменьшить масштаб развертывания, запустите следующую команду. Здесь мы сокращаем количество реплик до трех, но можно было задать и любое другое целое положительное число:

```
kubectl scale deployment packt --replicas=3
```

7. Если теперь запустить **get pods**, мы увидим, как лишние модули завершаются, пока их не останется три:

```
kubectl get pods -l app=packt
```

8. Прежде чем закончить разговор об императивных развертываниях, давайте очистим все ресурсы, которые мы создали до сих пор:

```
alexandru@debian:~$ kubectl delete service packt
service "packt" deleted
```

```
alexandru@debian:~$ kubectl delete deployment packt
deployment.apps "packt" deleted
```

```
alexandru@debian:~$ kubectl delete pod packt-web
pod "packt-web" deleted
```

9. Если теперь запросить все ресурсы в текущем пространстве имен, команда должна вывести сведения о кластере в исходном состоянии:

```
alexandru@debian:~$ kubectl get all
NAME                                TYPE             CLUSTER-IP   EXTERNAL-IP   PORT(S)    AGE
service/kubernetes                 ClusterIP        10.96.0.1    <none>        443/TCP    29h
```

В следующем подразделе мы рассмотрим, как декларативно развертывать ресурсы и приложения в кластере Kubernetes.

Декларативные развертывания

Декларативное развертывание опирается на файл манифеста. Такие файлы обычно представлены в формате YAML, и их код частично генерируется автоматически, а частично пишется вручную. Затем манифест развертывается с помощью команды `kubectl apply`:

```
kubectl apply -f файл_манифеста
```

Декларативное развертывание ресурсов в Kubernetes состоит из таких операций (они могут повторяться в цикле):

- создать файл манифеста;
- обновить манифест;
- проверить синтаксис манифеста;
- развернуть манифест.

Чтобы проиллюстрировать декларативную модель, мы снова развернем в кластере простое веб-приложение Hello World. Результат будет похож на то, чего мы добились императивным методом.

Давайте сначала создадим манифест для нашего развертывания.

Как создать манифест

Вспомните, какую команду мы запускали, когда создавали развертывание `packt` в интерактивном режиме (пока не запускайте `ee!`):

```
kubectl create deployment packt --image=nginxdemos/hello
```

Следующая команда смоделирует тот же процесс, не изменяя состояние системы:

```
alexandru@debian:~$ kubectl create deployment packt --image=nginxdemos/hello \
  --dry-run=client --output=yaml
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
```

```
labels:
  app: packt
  name: packt
...
```

Мы использовали такие дополнительные ключи:

<code>--dry-run=client</code>	Запустить команду в локальном клиентском окружении <code>kubectl</code> , не изменяя состояние системы
<code>--output=yaml</code>	Форматировать вывод команды в YAML

По выводу этой команды можно судить о том, какие изменения требуется внести в систему. Давайте перенаправим этот вывод в файл `packt.yaml`, который будет черновиком нашего манифеста развертывания:

```
kubectl create deployment packt --image=nginxdemos/hello \
  --dry-run=client --output=yaml > packt.yaml
```

Теперь у нас есть наш первый файл манифеста `packt.yaml`, который можно редактировать, чтобы приспособить к более сложным конфигурациям. Но для начала мы оставим его как есть и перейдем к следующему этапу декларативного развертывания.

Как проверить синтаксис манифеста

Перед тем как разворачивать манифест, стоит проверить его синтаксис, особенно если вы редактировали файл вручную. Синтаксические ошибки — не редкость, если вы редактируете сложные файлы YAML с несколькими уровнями отступов.

Следующая команда проверяет синтаксис манифеста `packt.yaml`:

```
kubectl apply -f packt.yaml --dry-run=client
```

Если ошибок нет, вы увидите такой результат:

```
deployment.apps/packt created (dry run)
```

Если обнаружены ошибки, придется исправить файл манифеста, прежде чем его можно будет разворачивать. В нашем манифесте ошибок нет, так что давайте приступим к развертыванию.

Как развернуть манифест

Чтобы развернуть манифест `packt.yaml`, запустите такую команду:

```
alexandru@debian:~$ kubectl apply -f packt.yaml
deployment.apps/packt created
```

Следующая команда позволяет проверить, что развернуты нужные ресурсы:


```
alexandru@debian:~$ kubectl get all -l app=packt
```

NAME	READY	STATUS	RESTARTS	AGE
pod/packt-579bb9c999-9ks6x	1/1	Running	0	12s

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/packt	1/1	1	1	12s

NAME	DESIRED	CURRENT	READY	AGE
replicaset.apps/packt-579bb9c999	1	1	1	12s

Этот результат показывает, что ресурсы развертывания `packt`, которые были созданы в декларативном режиме, запущены и работают.

Как предоставить доступ к развертыванию с помощью службы

Мы повторим предыдущее упражнение, в котором открывали доступ к развертыванию для внешнего мира, но на сей раз создадим, проверим и развернем манифест соответствующей службы (`packt-svc.yaml`). Для краткости просто перечислим нужные команды:

1. Создайте файл манифеста `packt-svc.yaml` для службы, которая предоставляет доступ к нашему развертыванию `packt`:

```
kubectl expose deployment packt \
  --port=80 \
  --target-port=80 \
  --type=NodePort \
  --dry-run=client --output=yaml > packt-svc.yaml
```

(Мы уже рассматривали аналогичную команду в разделе «Императивные развертывания».)

2. Проверьте манифест развертывания службы:

```
kubectl apply -f packt-svc.yaml --dry-run=client
```

3. Если проверка прошла успешно, разверните манифест:

```
kubectl apply -f packt-svc.yaml
```

4. Просмотрите текущее состояние ресурсов `packt`:

```
kubectl get all -l app=packt
```

Теперь к выводу добавилась конечная точка службы `service/packt`, которая прослушивает порт `31380`:

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service/packt	NodePort	10.106.74.34	<none>	80:31380/TCP	17s

5. Теперь с помощью обычного браузера, `curl` или `Lynx` можно обратиться к нашему приложению, запросив любой узел кластера на порте `31380`. Давайте запросим управляющий узел (`k8s-cp1`), введя в браузере адрес `http://192.168.122.104:31380`:



Рис. 16.6. Доступ к конечной точке приложения `packt`

Если нужно изменить конфигурацию ресурса в приложении разворачивания, можно обновить соответствующий манифест и переразвернуть его. В следующем подразделе мы модифицируем разворачивание, чтобы приспособить его к масштабированию.

Как обновить манифест

Предположим, что наше приложение принимает много запросов и мы хотим добавить в разворачивание больше подов, чтобы справиться с трафиком. Для этого в манифесте `packt.yaml` нужно изменить параметр `spec.replicas`:

1. Откройте `packt.yaml` в текстовом редакторе и найдите такой раздел конфигурации:

```
spec:
  replicas: 1
```

Чтобы добавить поды приложения в набор реплик под управлением разворачивания `packt`, увеличьте значение (например, до 10):

```
spec:
  replicas: 10
```

2. Сохраните файл манифеста и разверните его заново с помощью такой команды:

```
alexandru@debian:~$ kubectl apply -f packt.yaml
deployment.apps/packt configured
```

Вывод показывает, что разворачивание `packt` успешно перенастроено.

3. Если теперь запросить ресурсы разворачивания `packt` в кластере, вы увидите, что новые поды запущены:

```
alexandru@debian:~$ kubectl get all -l app=packt
```

NAME	READY	STATUS	RESTARTS	AGE
pod/packt-579bb9c999-9ks6x	1/1	Running	0	39s
pod/packt-579bb9c999-5pw6q	1/1	Running	0	14m
pod/packt-579bb9c999-9ks6x	1/1	Running	0	39s
pod/packt-579bb9c999-cvptf	1/1	Running	0	39s
pod/packt-579bb9c999-d7kmj	1/1	Running	0	39s
pod/packt-579bb9c999-fpckr	1/1	Running	0	39s
pod/packt-579bb9c999-gr29g	1/1	Running	0	39s
pod/packt-579bb9c999-k94jz	1/1	Running	0	39s
pod/packt-579bb9c999-m55c8	1/1	Running	0	39s
pod/packt-579bb9c999-p492x	1/1	Running	0	39s

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
service/packt	NodePort	10.106.74.34	<none>	80:31380/TCP	8m38s

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
deployment.apps/packt	10/10	10	10	14m

NAME	DESIRED	CURRENT	READY	AGE
replicaset.apps/packt-579bb9c999	10	10	10	14m

Попробуйте протестировать новую конфигурацию в масштабируемой среде и проверить балансировку нагрузки, как описывалось ранее в разделе «Как масштабировать развертывания приложений».

- Теперь давайте уменьшим наше развертывание до трех подов, но на этот раз обновим соответствующий манифест на лету с помощью такой команды:

```
kubectl edit deployment packt
```

Она откроет стандартный текстовый редактор (в нашем случае **vi**), в котором можно внести нужное изменение:

```
spec:
  progressDeadlineSeconds: 600
  replicas: 3
  revisionHistoryLimit: 10
```

- После того как вы сохраните файл и выйдете из редактора, вы увидите сообщение о том, что развертывание **packt** было обновлено:

```
deployment.apps/packt edited
```

Обратите внимание: все, что мы модифицировали на лету с помощью **kubectl edit**, не отражается в манифесте развертывания **packt.yaml**, однако соответствующие изменения конфигурации сохраняются в кластере.

- Чтобы проверить обновленное развертывание, запустите такую команду:

```
alexandru@debian:~$ kubectl get deployment packt
```

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
packt	3/3	3	3	18m

Теперь вывод показывает, что в нашем разворачивании запущены только три пода.

7. Прежде чем завершить работу, давайте снова очистим ресурсы, чтобы вернуть кластер в состояние по умолчанию:

```
kubectl delete service packt
kubectl delete deployment packt
```

Мы рассмотрели, как использовать Kubernetes на физическом оборудовании, а в следующем разделе кратко расскажем о соответствующих облачных службах.

Как запускать Kubernetes в облаке

Многие публичные облачные провайдеры предлагают управляемые службы Kubernetes. Самые распространенные решения на момент написания этой книги — **Amazon Elastic Kubernetes Service (EKS)**, **Azure Kubernetes Service (AKS)** и **Google Kubernetes Engine (GKE)**. Здесь мы не будем их подробно рассматривать, но предлагаем обратить внимание на ссылки в разделе «Дополнительная литература» в конце этой главы, чтобы получить дополнительную информацию.

Отметим, что мы только в общих чертах познакомились с тем, как разворачивать кластеры Kubernetes и управлять ими. Тем не менее мы проделали важную работу, когда развернули свои первые кластеры на локальном компьютере. Мы приближаемся к концу этой темы, но надеемся, что вы продолжите оттачивать свои навыки на практике, разворачивая и масштабируя приложения с помощью Kubernetes.

Итоги

Мы начали эту главу с высокоуровневого обзора архитектуры Kubernetes и объектной модели API, где представили ключевые ресурсы кластера, такие как поды, разворачивания и службы. После этого мы взялись за довольно сложную задачу — создать локальный кластер Kubernetes с нуля с помощью виртуальных машин. Мы изучили различные инструменты командной строки, которые позволяют управлять ресурсами кластера Kubernetes на локальном узле. Наконец, мы узнали, как разворачивать и масштабировать приложения в Kubernetes в императивном и декларативном режиме.

Мы уверены, что начинающие администраторы Linux получают большую пользу от материала этой главы и смогут уверенно управлять ресурсами в гибридных облаках и локальных распределенных средах, масштабировать разворачивания приложений и работать с инструментами командной строки. Предполагаем, что этот материал также поможет опытным системным администраторам освежить свои знания и навыки.

Эта глава получилась довольно длинной, хотя мы затронули только самые основные понятия Kubernetes. Мы рекомендуем вам изучить ресурсы, которые

перечислены в разделе «Дополнительная литература», и укрепить свои знания о таких важных аспектах сред Kubernetes, как сетевые технологии, безопасность и масштабирование.

В следующей главе мы останемся в сфере развертывания приложений и рассмотрим систему управления конфигурациями **Ansible**, которая ускоряет доставку приложений в локальных и облачных средах.

Вопросы

Вот несколько вопросов, которые помогут вам закрепить основные темы этой главы:

1. Перечислите основные службы управляющего узла Kubernetes. Чем отличаются рабочие узлы от управляющего?
2. Какая команда позволяет загрузить кластер Kubernetes?
3. В чем разница между императивными и декларативными развертываниями в Kubernetes?
4. Как выглядит команда `kubectl`, которая развертывает под? А как насчет команды, которая создает развертывание (объект `Deployment`)?
5. С помощью какой команды `kubectl` можно получить доступ к командной оболочке внутри контейнера пода?
6. С помощью какой команды `kubectl` можно запросить все ресурсы, которые связаны с развертыванием?
7. Как масштабировать развертывание в Kubernetes? Перечислите различные команды, которые выполняют эту задачу.
8. Как удалить все ресурсы, которые связаны с развертыванием в Kubernetes?

Дополнительная литература

Эти источники помогут вам расширить свои знания о Kubernetes.

Документация по Kubernetes: kubernetes.io/docs.

Шпаргалка по `kubectl`: kubernetes.io/docs/reference/kubectl/quick-reference:

- Видеокурс «Kubernetes and Docker: The Container Masterclass» от компании Cerulean Canvas.
- Джиджи Сайфан (Gigi Sayfan). «Mastering Kubernetes», Third Edition¹.

Также предлагаем вашему вниманию полезные ссылки о том, как работать с развертываниями Kubernetes в средах Azure, Amazon и Google:

¹ Сайфан Дж. «Осваиваем Kubernetes. Оркестрация контейнерных архитектур». — СПб., изд-во «Питер».

- Amazon Elastic Kubernetes Service:
 - Главная страница документации: docs.aws.amazon.com/eks.
 - Пример разворачивания приложения под Linux: docs.aws.amazon.com/eks/latest/userguide/sample-deployment.html.
- Azure Kubernetes Service:
 - Главная страница AKS: azure.microsoft.com/products/kubernetes-service.
 - Учебное пособие по созданию кластера AKS: learn.microsoft.com/azure/aks/tutorial-kubernetes-deploy-cluster.
 - Как развернуть кластер AKS с помощью портала Azure: learn.microsoft.com/azure/aks/learn/quick-kubernetes-deploy-portal.
- Google Kubernetes Engine:
 - Главная страница GKE: cloud.google.com/kubernetes-engine.
 - Развертывание в GKE: cloud.google.com/build/docs/deploying-builds/deploy-gke.
 - Как развернуть приложение в кластере GKE: cloud.google.com/kubernetes-engine/docs/deploy-app-cluster.

17

Как управлять инфраструктурой и автоматизацией с помощью Ansible

Если ваша повседневная работа по системному администрированию или разработке связана с утомительными и повторяющимися операциями, инструмент **Ansible** поможет вам справиться с задачами и сэкономить драгоценное время. С помощью Ansible можно автоматизировать процессы, которые **настраивают программное обеспечение, управляют конфигурациями и развертывают приложения**. Изначально Ansible разработал Майкл ДеХан в 2012 году, а в 2015 году проект приобрела компания Red Hat, которая теперь поддерживает его как программный продукт с открытым исходным кодом.

В этой главе вы познакомитесь с фундаментальными понятиями Ansible, а также увидите много практических примеров. В частности, мы рассмотрим такие темы:

- Архитектура и управление конфигурацией Ansible.
- Как установить Ansible.
- Как работать с Ansible.

Технические требования

Материал этой главы подразумевает, что вы в целом знакомы с командной строкой Linux и умеете пользоваться текстовым редактором. Если у вас есть навыки продвинутого пользователя Linux, вам будет проще понять некоторые тонкости практических примеров из этой главы.

Для упражнений мы рекомендуем создать такую же учебную среду, как та, которую используем мы. Чтобы ее воспроизвести, вам понадобится процессор, у которого не менее шести физических и не менее шести виртуальных ядер (то есть всего хотя бы 12). Четырехъядерного процессора с гиперпоточностью недостаточно. Кроме того, на всех узлах должен быть установлен OpenSSH. Все необходимые инструкции вы найдете в разделе «Учебная среда для Ansible»

далее в этой главе. Даже если вы не будете настраивать учебную среду, вы все равно сможете извлечь пользу из подробных объяснений, которые сопровождают практические примеры в этой главе.

Итак, давайте для начала рассмотрим вводные понятия, связанные с Ansible.

Архитектура и управление конфигурацией Ansible

Во введении к этой главе мы отметили одно из важнейших качеств Ansible: это инструмент для того, чтобы автоматизировать рабочие процессы. Практически любую задачу системного администрирования Linux можно автоматизировать с помощью Ansible. Интерфейс командной строки Ansible позволяет вызывать простые команды, чтобы изменить желаемое состояние системы. Обычно с помощью Ansible выполняют задачи на удаленном узле или группе узлов.

Давайте воспользуемся классической иллюстрацией **управления пакетами**. Предположим, что вы управляете инфраструктурой, в которую входит группа веб-серверов, и планируете установить последнюю версию серверного приложения (nginx или Apache) на все эти узлы. Один из способов решить задачу — зайти по SSH на каждый узел и запустить нужные команды оболочки, чтобы установить новый пакет. Если у вас много узлов, это превратится в довольно муторную работу. Вы можете возразить, что ее можно автоматизировать, если написать соответствующий сценарий командной оболочки. Это осуществимо, но тогда появятся новые задачи: сопровождать сценарий, исправлять возможные ошибки и добавлять новые функции, когда инфраструктура разрастется.

Скорее всего, в какой-то момент вам понадобится управлять пользователями или базами данных, а может быть, настраивать конфигурацию сети на нескольких узлах. В результате окажется, что вам нужен многофункциональный инструмент вроде швейцарского ножа, который вы бы предпочли получить в готовом виде, а не разрабатывать самостоятельно. Вот тут-то и пригодится Ansible, который поддерживает огромное количество модулей — практически для любой задачи системного администрирования, которую только можно себе представить. Благодаря этому Ansible позволяет удаленно настраивать, запускать или развертывать административные задачи с минимальными усилиями, причем вполне безопасно и эффективно.

После этих предварительных соображений давайте вкратце рассмотрим архитектуру Ansible.

Введение в архитектуру Ansible

Основной фреймворк Ansible написан на языке Python. Сразу отметим, что Ansible имеет **безагентную** архитектуру: он работает на **управляющем узле**, который запускает команды на удаленных узлах, и на них не требуется установ-

ливать удаленную конечную точку или службу для связи с управляющим узлом. Единственное требование — соединение по SSH с каждым управляемым узлом. Если на узле не установлен фреймворк Python, Ansible сможет только запускать сценарии оболочки и обычные команды, однако в подавляющем большинстве серверных платформ Python уже установлен по умолчанию.

С одного узла Ansible может управлять целым парком удаленных узлов, обращаясь к ним через защищенные соединения SSH. Вот логическая схема инфраструктуры под управлением Ansible:

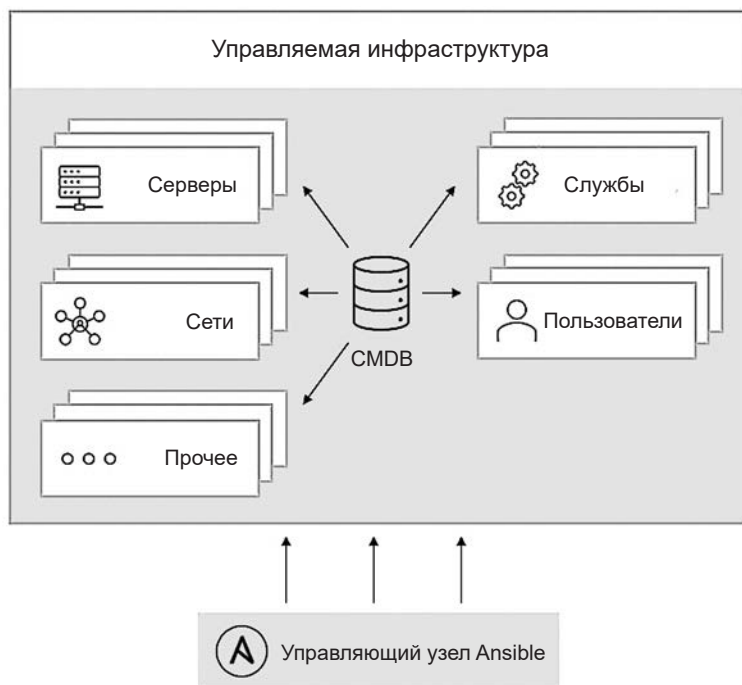


Рис. 17.1. Логическая схема инфраструктуры под управлением Ansible

В среды эксплуатации корпоративного класса обычно входит **CMDB** (configuration management database, база данных управления конфигурацией), с помощью которой компании организывают активы своей компьютерной инфраструктуры. Примеры таких активов — серверы, сети, службы и пользователи. Хотя сама CMDB не входит в архитектуру Ansible, она описывает, какие активы есть в управляемой инфраструктуре и как они связаны между собой. Эти сведения можно использовать, чтобы создать **опись Ansible**.

Опись (inventory) — это локальное хранилище на управляющем узле Ansible. Обычно это файл в формате **INI** или **YAML**, в котором описаны управляемые **узлы** или **группы** узлов. Опись может автоматически генерироваться на основе CMDB, а может создаваться вручную системным администратором.

Теперь давайте подробнее рассмотрим высокоуровневую архитектуру Ansible, которая изображена на следующей диаграмме:

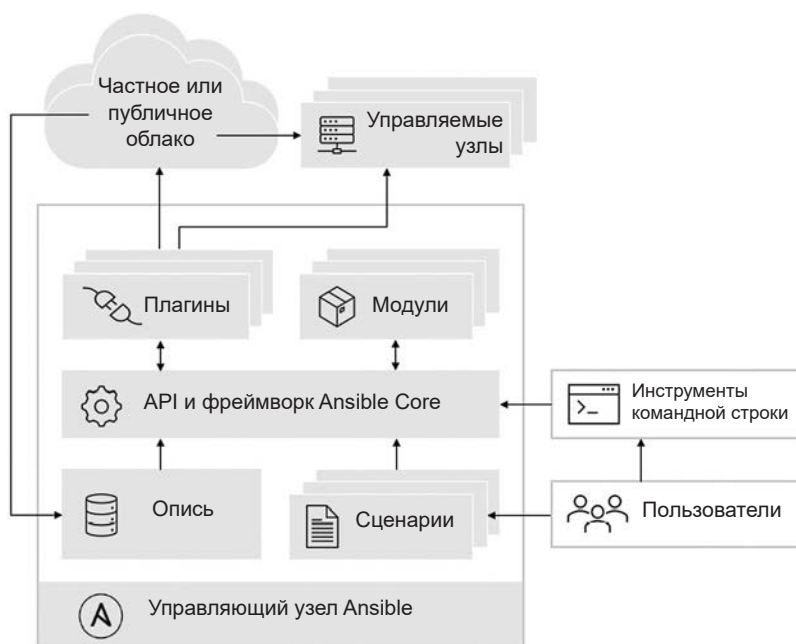


Рис. 17.2. Архитектура Ansible

Здесь показан управляющий узел Ansible, который взаимодействует с управляемыми узлами в частной или публичной облачной среде. Вот основные компоненты этой архитектуры:

- **API и фреймворк Ansible Core.** Ключевые библиотеки, которые инкапсулируют основную функциональность Ansible. Фреймворк Ansible Core написан на Python.
- **Плагины.** Дополнительные библиотеки, которые расширяют функциональность Core, в том числе:
 - **плагины для подключения** (например, к облачной среде);
 - **плагины для тестирования**, которые проверяют те или иные данные ответа;
 - **плагины обратного вызова**, которые позволяют реагировать на события.
- **Модули** инкапсулируют различные функции, которые выполняются на управляемых узлах. Например, с помощью модуля `user` можно управлять пользователями, а с помощью модуля `package` — пакетами программного обеспечения.
- **Опись.** Этот файл в формате INI или YAML описывает узлы и группы узлов, к которым обращаются команды и сценарии Ansible.

- **Сценарии («плейбуки»).** Эти файлы с инструкциями Ansible описывают набор задач, которые передаются управляемым узлам.
- **Частные или публичные облака.** Управляемая инфраструктура, которая размещается локально или в различных облачных средах (например, AWS или Azure).
- **Управляемые узлы.** Узлы, на которых выполняются команды и сценарии Ansible.
- **Инструменты командной строки** Ansible — например, `ansible`, `ansible-playbook` и `ansible-doc`.
- **Пользователи.** Администраторы, продвинутые пользователи и автоматизированные пользовательские процессы, которые выполняют команды или сценарии Ansible.

Теперь, когда вы в целом понимаете архитектуру Ansible, давайте поговорим о том, почему он стал отличным инструментом для того, чтобы автоматизировать административные задачи, и узнаем, что такое **управление конфигурацией**.

Что такое управление конфигурацией

В прежние времена системные администраторы обычно управляли относительно небольшим количеством серверов и занимались повседневными административными задачами с помощью удаленной командной оболочки на каждом узле. Чтобы выполнять относительно простые операции — например, копировать файлы, обновлять пакеты программного обеспечения и управлять пользователями, — можно было легко писать сценарии и использовать их повторно. Но бурное развитие интернета привело к тому, что современные локальные и облачные инфраструктуры существенно разрослись. Чтобы их сопровождать, больше не хватает возможностей одного администратора с горсткой самодельных сценариев. Именно здесь на помощь приходит управление конфигурацией.

При управлении конфигурацией управляемые группы и активы объединяются в логические категории по определенным критериям, как показано на рис. 17.1. Управление всеми активами, кроме узлов, в конечном счете сводится к тому, чтобы запускать определенные задачи на серверах, где размещены эти активы. **Манифест управления конфигурацией** — это файл описи Ansible, который управляет этими узлами и активами. Таким образом, Ansible становится оконечной точкой управления конфигурацией.

Ansible позволяет запускать одиночные команды, которые выполняют те или иные задачи, но гораздо более эффективный путь — управлять конфигурацией с помощью **сценариев** Ansible (так называемых плейбуков), которые позволяют запускать множество задач, направленных на различные подсистемы целевой платформы, на скольких угодно узлах. Обычной практикой автоматизации вычислительной инфраструктуры стало планировать запуск `ansible-playbook`, чтобы регулярно обслуживать конфигурацию и управлять ею. Мы обсудим сценарии Ansible позже в этой главе.

Когда задачи Ansible для определенной целевой системы запускаются многократно (или повторяются по расписанию), есть повод опасаться, что повторяющиеся операции могут некорректно изменить состояние системы относительно желаемого. Этот вопрос подводит нас к одному из важнейших аспектов управления конфигурацией — **идемпотентности** ее изменений. Давайте разберемся, что это значит.

Что такое идиempotentные операции

В управлении конфигурацией операция считается **идемпотентной**, если ее многократное выполнение приводит к тому же результату, что и однократное. В этом смысле Ansible — идиempotentный инструмент управления конфигурацией.

Давайте объясним, как это работает. Предположим, у нас есть задача Ansible, которая создает учетную запись. Когда задача запускается в первый раз, в результате возникает новая учетная запись. А если после этого запустить задачу повторно, Ansible просто не выполнит никаких операций. Если бы не было идиempotentности, то повторные запуски задачи приводили бы к ошибкам из-за того, что мы пытаемся создать пользователя, который уже существует.

Отметим, что Ansible — не единственный инструмент управления конфигурацией: на рынке представлены еще **Chef**, **Puppet**, **SaltStack** и другие решения. Большинство из этих продуктов были приобретены более крупными компаниями: например, SaltStack принадлежит VMware. Многие специалисты объясняют успех Ansible тем, что Red Hat открыла исходный код этой платформы. На момент написания этой книги Ansible выглядит самым успешным инструментом управления конфигурациями. Он удобен в использовании, хорошо масштабируется и предлагает приемлемую стоимость лицензий, если вы разворачиваете его в корпоративном сегменте.

Ознакомившись с вводными понятиями, давайте перейдем к практике и установим Ansible в системе Linux.

Как установить Ansible

В этом разделе мы покажем, как установить Ansible на **управляющий узел**. В Linux это можно сделать несколькими способами:

- с помощью системы управления пакетами для конкретной платформы (например, **apt** в Ubuntu/Debian);
- с помощью **pip** — системы управления пакетами Python.

Сообщество Ansible рекомендует второй способ, потому что он обеспечивает самую последнюю стабильную версию Ansible. В примерах из этого раздела мы будем использовать Ubuntu. Полное руководство о том, как установить Ansible

во всех основных операционных системах, можно найти в официальной документации: docs.ansible.com/ansible/latest/installation_guide/intro_installation.html.

На управляющем узле для Ansible нужен Python, поэтому перед тем, как устанавливать Ansible, стоит убедиться, что в системе есть Python.

Начнем с того, что установим Ansible в Ubuntu.

Как установить Ansible в Ubuntu

В Ubuntu 22.04 LTS по умолчанию установлен Python 3. Чтобы установить Ansible, выполните такие действия:

1. Сначала проверьте, какая версия Python 3 установлена в системе:

```
python3 --version
```

На нашем компьютере под управлением Ubuntu версия Python выглядит так:

```
Python 3.10.12
```

Если на вашей машине нет Python 3, его можно установить с помощью такой команды:

```
sudo apt install -y python3
```

2. Теперь можно приступить к установке Ansible. Чтобы получить самые свежие пакеты Ansible с помощью `apt`, надо добавить в систему репозиторий **Ansible PPA**. Давайте сначала обновим текущий кэш пакетов `apt`:

```
sudo apt update
```

3. Далее добавьте репозиторий Ansible PPA:

```
sudo apt install -y software-properties-common1  
sudo apt-add-repository --update ppa:ansible/ansible -y2
```

4. Наконец, установите пакет Ansible:

```
sudo apt install ansible -y
```

5. Когда установка завершится, можно проверить текущую версию пакета:

```
ansible --version
```

¹ В современных версиях Ubuntu и многих других дистрибутивах пакет `software-properties-common` установлен по умолчанию. — *Примеч. ред.*

² Обратите внимание, что устанавливать Ansible с помощью `apt` официально не рекомендуется, и этот способ может вызвать проблемы: например, `apt` может вывести сообщение о том, что репозиторий `ppa:ansible/ansible` не является доверенным, и из него не удастся устанавливать пакеты. Официальная документация рекомендует устанавливать Ansible с помощью `pip` (см. далее) или `pipx`. — *Примеч. ред.*

Вывод состоит из нескольких строк, и в первой из них указана версия Ansible Core, например:

```
ansible [core 2.18.1]
```

Далее давайте посмотрим, как установить Ansible с помощью `pip`.

Как установить Ansible с помощью `pip`

Прежде чем устанавливать Ansible с помощью `pip`, убедитесь, что в системе установлен Python. (В предыдущем разделе мы рассказывали, как это сделать.) Кроме того, рекомендуем удалить все версии этого Ansible, которые были установлены ранее (например, с помощью `apt`); в противном случае `pip` может повести себя некорректно. Выполните такие действия:

1. Удалите все версии Ansible, которые были установлены с помощью системы управления пакетами (например, `apt` или `yum`). В частности, чтобы удалить Ansible в Ubuntu, запустите такую команду:

```
sudo apt remove -y ansible
```

2. Убедитесь, что `pip` установлен. Эта команда должна вывести его текущую версию:

```
python3 -m pip --version
```

Если `pip` не установлен, вы увидите сообщение об ошибке:

```
/usr/bin/python3: No module named pip
```

В этом случае сначала загрузите установщик `pip`¹, а затем запустите его:

```
curl -s https://bootstrap.pypa.io/get-pip.py -O  
python3 get-pip.py
```

3. Теперь можно установить Ansible:

```
python3 -m pip install --user ansible
```

Обратите внимание, что благодаря ключу `--user` Ansible устанавливается только для текущего пользователя. Если вы хотите установить Ansible *глобально* в системе, команда будет выглядеть так:

```
sudo python3 -m pip install ansible
```

4. Возможно, после установки вам придется выйти и снова войти в систему, прежде чем использовать Ansible. Проверить версию Ansible можно с помощью такой команды:

```
ansible --version
```

В нашем случае вывод начинается с такой строки:

```
ansible [core 2.18.1]
```

¹ Рекомендуется устанавливать `pip` при помощи пакетного менеджера, например: `dnf install python3-pip`. — *Примеч. ред.*

Как видите, с помощью `pip` можно получить самую свежую версию Ansible, поэтому данный метод установки рекомендуется официально.

Установив Ansible на управляющий узел, давайте рассмотрим несколько практических примеров его использования.

Как работать с Ansible

В этом разделе мы будем активно использовать инструменты командной строки Ansible, чтобы выполнять различные задачи по управлению конфигурацией. Мы будем выполнять практические упражнения в учебной среде, которую настоятельно рекомендуем вам воспроизвести в своей системе, чтобы получить полноценный опыт управления конфигурацией.

Вот общий план этого раздела:

- Учебная среда для Ansible.
- Как настроить Ansible.
- Как использовать разовые команды Ansible.
- Как работать со сценариями Ansible.
- Как применять шаблоны в Jinja2.
- Как использовать роли Ansible.

Начнем с обзора учебной среды.

Учебная среда для Ansible

В дальнейших упражнениях с виртуальными машинами мы будем использовать гипервизор **KVM**, но подойдет и любой другой. В главе 11 «Как работать с виртуальными машинами» подробно рассказывалось, как создавать виртуальные машины в Linux. Чтобы имитировать реальную инфраструктуру управления конфигурацией, мы развернули такие виртуальные машины под управлением Ubuntu Server LTS:

neptune	Управляющий узел Ansible
ans-web1 ans-web2	Веб-серверы
ans-db1 ans-db2	Серверы баз данных

На всех виртуальных машинах установлены серверные компоненты по умолчанию. На каждом узле мы создали стандартную учетную запись с именем `packt` и настроили для нее доступ по SSH. У каждой виртуальной машины будет два виртуальных процессора, 2 Гбайта оперативной памяти и 20 Гбайт дискового пространства.

Теперь давайте вкратце опишем, как настроить эти виртуальные машины, начиная с управляемых узлов.

Как настроить управляемые узлы

Чтобы полностью обеспечить доступ к управлению конфигурацией с управляющего узла Ansible, на управляемых узлах должны быть выполнены два требования:

- установлен и запущен сервер OpenSSH;
- установлен Python.

Как говорилось в разделе «Технические требования», мы предполагаем, что на ваших узлах настроен OpenSSH. Чтобы установить Python, выполните соответствующую процедуру из раздела «Как установить Ansible».

На управляемых узлах не требуется устанавливать Ansible.

Чтобы задать имя узла для каждой виртуальной машины, запустите команду `hostnamectl`. Например, так она выглядит для узла `ans-web1`:

```
sudo hostnamectl set-hostname ans-web1
```

На управляемых узлах имеет смысл отключить запрос пароля для `sudo`, чтобы было легче повышать привилегии, когда запускаются автоматические сценарии. Если не внести это изменение, система будет требовать пароль, чтобы удаленно выполнять команды Ansible.

Чтобы отключить запрос пароля, отредактируйте конфигурацию `sudo` с помощью такой команды:

```
sudo visudo
```

Добавьте следующую строку и сохраните конфигурационный файл. Вместо `packt` подставьте свое имя пользователя:

```
packt ALL=(ALL) NOPASSWD:ALL
```

Эту настройку понадобится применить на каждом управляемом узле.

Далее мы посмотрим, как настроить управляющий узел Ansible.

Как настроить управляющий узел Ansible

Управляющий узел Ansible (`neptune`) будет взаимодействовать с **управляемыми узлами** (`ans-web1`, `ans-web2`, `ans-db1` и `ans-db2`) с помощью команд и сценариев Ansible. Для удобства в наших примерах мы будем обращаться к управляемым узлам по именам, а не по IP-адресам. Чтобы добиться этого, добавьте такие записи в файл `/etc/hosts` на управляющем узле (`neptune`):


```
127.0.0.1 neptune localhost
192.168.122.70 ans-web1
192.168.122.147 ans-web2
192.168.122.254 ans-db1
192.168.122.25 ans-db2
```

Не забудьте изменить IP-адреса в соответствии с сетевым окружением ваших виртуальных машин.

Наконец, давайте настроим аутентификацию по ключу SSH между управляющим и управляемыми узлами Ansible.

Как настроить аутентификацию по ключу SSH

Управляющий узел Ansible связывается с управляемыми по протоколу SSH. Аутентификация с открытым ключом позволяет получить удаленный доступ по SSH без необходимости вводить пароли пользователей. Чтобы настроить аутентификацию, выполните следующие команды на управляющем узле (*neptune*).

Чтобы создать защищенную пару ключей, запустите `ssh-keygen` и следуйте подсказкам по умолчанию:

```
ssh-keygen
```

Сгенерировав пару ключей, скопируйте открытый ключ на каждый управляемый узел. Эту процедуру придется выполнить по отдельности на каждом узле, где сперва понадобится аутентифицироваться с помощью пароля удаленного пользователя `packt`. Например, так выглядит нужная команда для первого управляемого узла (*ans-web1*):

```
ssh-copy-id -i ~/.ssh/id_rsa.pub packt@ans-web1
```

Повторите эту операцию для остальных узлов (*ans-web2*, *ans-db1*, *ans-db2*). Если все прошло успешно, то теперь вы можете подключаться по SSH с управляющего узла Ansible (*neptune*) к любому из управляемых узлов, не вводя пароль. Например, чтобы подключиться к *ans-web1*, можно запустить такую команду:

```
ssh packt@ans-web1
```

В результате вы переключитесь в терминал удаленного сервера *ans-web1*. Прежде чем выполнять следующие шаги, убедитесь, что вы вернулись на управляющий узел Ansible (*neptune*).

Теперь давайте настроим Ansible на управляющем узле.

Как настраивать Ansible

В этом разделе рассматриваются основные понятия конфигурации Ansible, которые относятся к **конфигурационному файлу** и **описи**. С помощью конфи-

гурационного файла и параметров в нем можно настраивать поведение Ansible: например, повышение привилегий, время ожидания соединения и путь к файлу описи по умолчанию. Опись определяет управляемые узлы, выступая в качестве CMDB для Ansible.

Давайте рассмотрим конфигурационный файл Ansible.

Как создать конфигурационный файл Ansible

Напомним команду, которая предоставляет полезную информацию о нашей среде Ansible, включая текущий конфигурационный файл:

```
packt@neptune:~$ ansible --version
ansible [core 2.15.4]
  config file = /etc/ansible/ansible.cfg
  configured module search path = ['/home/packt/.ansible/plugins/modules', ... ]
  ansible python module location = /home/packt/.local/lib/python3.10/...
  ansible collection location = /home/packt/.ansible/collections:...
  executable location = /home/packt/.local/bin/ansible
  python version = 3.10.12 (main, Jun 11 2023, 05:26:28) [GCC 11.4.0] ...
  jinja version = 3.0.3
  libyaml = True
```

По умолчанию конфигурационный файл Ansible находится в расположении `/etc/ansible/ansible.cfg`. Как вы наверняка догадались, он имеет *глобальную область видимости*, то есть используется по умолчанию для всех задач Ansible.

Посмотрим, какие проблемы с этим связаны и как их решить:

- Что делать, если на одном управляющем узле несколько пользователей запускают задачи Ansible? Интуиция подсказывает, что у каждого из них может быть собственная конфигурация. Чтобы учесть ее, Ansible заглядывает в домашний каталог пользователя и ищет там файл `~/.ansible.cfg`. В этом можно убедиться, если создать фиктивный конфигурационный файл в нашем домашнем каталоге:

```
touch ~/.ansible.cfg
```

Если теперь снова вызвать команду `ansible --version`, она будет указывать на другой конфигурационный файл:

```
packt@neptune:~$ ansible --version
ansible [core 2.15.4]
  config file = /home/packt/.ansible.cfg
  ...
```

Другими словами, пользовательский конфигурационный файл `~/.ansible.cfg` имеет приоритет над глобальным файлом `/etc/ansible/ansible.cfg`.

- Теперь предположим, что пользователь (в нашем случае `packt`) работает с несколькими проектами Ansible, из которых одни управляют локальными узлами, а другие взаимодействуют с публичными облачными ресурсами. Нам снова могут понадобиться разные наборы параметров конфигурации (на-

пример, время ожидания соединения и файл описи). Чтобы учесть и этот сценарий, Ansible ищет файл `./ansible.cfg` в текущем каталоге.

Давайте создадим каталог `~/ansible/`, а в нем фиктивный файл `ansible.cfg`:

```
mkdir ~/ansible
touch ~/ansible/ansible.cfg
```

Если перейти в каталог `~/ansible` и там вызвать команду `ansible --version`, мы увидим новый конфигурационный файл:

```
packt@neptune:~$ cd ansible

packt@neptune:~/ansible$ ansible --version
ansible [core 2.15.4]
  config file = /home/packt/ansible/ansible.cfg
  ...
```

Каталог проекта может быть каким угодно, не обязательно `/home/packt/ansible`. Если в текущем каталоге есть конфигурационный файл, то Ansible отдает приоритет ему, а не файлу `~/ansible.cfg` в домашнем каталоге.

- Наконец, может понадобиться, чтобы Ansible обращался к определенному конфигурационному файлу, который не зависит от текущего каталога или каталога, из которого запускаются команды Ansible. Это полезно, когда вы тестируете разовые конфигурации и не хотите изменять основной конфигурационный файл. В этом случае можно задать переменную окружения `ANSIBLE_CONFIG`, к которой Ansible обращается, чтобы найти путь к конфигурационному файлу.

Допустим, мы находимся в каталоге проекта `~/ansible`, где уже расположен локальный конфигурационный файл `ansible.cfg`. Давайте создадим еще один фиктивный файл `test.cfg`:

```
cd ~/ansible
touch test.cfg
```

Теперь зададим переменную окружения `ANSIBLE_CONFIG` и убедимся, что после этого Ansible будет считывать конфигурацию из `test.cfg`, а не из `ansible.cfg`:

```
packt@neptune:~$ ANSIBLE_CONFIG=test.cfg ansible --version
ansible [core 2.15.4]
  config file = /home/packt/ansible/ansible.cfg
  ...
```

Обратите внимание, что конфигурационный файл должен иметь расширение `.cfg`, иначе Ansible его проигнорирует.

Таким образом, Ansible ищет конфигурационный файл в таком порядке (от более приоритетного к менее приоритетному):

1. Переменная среды `ANSIBLE_CONFIG`.
2. Файл `ansible.cfg` в текущем каталоге.

3. Файл `~/.ansible.cfg` в домашнем каталоге текущего пользователя.
4. Файл `/etc/ansible/ansible.cfg`.

В наших примерах будет фигурировать конфигурационный файл `ansible.cfg` в локальном каталоге проекта (`~/ansible`). Давайте создадим этот файл и пока оставим его пустым:

```
mkdir ~/ansible
cd ~/ansible
touch ansible.cfg
```

В оставшейся части этой главы мы будем запускать все команды Ansible, кроме специально отмеченных, из каталога `~/ansible`.

Для всех параметров, которые мы явно не определим в конфигурационном файле, Ansible задаст системные значения по умолчанию. Одним из параметров будет путь к файлу описи. Но сначала нужно создать саму опись, и в следующем подразделе мы покажем, как это сделать.

Как создать опись Ansible

Опись Ansible — это текстовый файл в формате **INI** или **YAML**, который описывает управляемые узлы. Простейшая опись может представлять собой одноуровневый список имен или IP-адресов узлов, но Ansible также позволяет организовывать узлы в **группы**. Файлы описи Ansible могут быть **статическими** или **динамическими** в зависимости от того, как они создаются и обновляются — вручную или автоматически. Для начала мы поработаем со статической описью.

Напоминаем, что в нашей демонстрационной среде есть два веб-сервера (`ans-web1`, `ans-web2`) и два сервера баз данных (`ans-db1`, `ans-db2`). Давайте создадим внутри каталога `~/ansible` файл `hosts` в формате INI и внесем в него такой код:

```
[webservers]
ans-web1
ans-web2

[databases]
ans-db1
ans-db2
```

Мы разделили наши узлы на две группы, названия которых заключены в квадратные скобки: `[webservers]` (веб-серверы) и `[databases]` (серверы баз данных). Как уже говорилось, группы служат для того, чтобы логически организовывать узлы на основе тех или иных критериев. Один и тот же узел может входить в несколько групп. Имена групп чувствительны к регистру, всегда начинаются с буквы и не должны содержать дефисов (-) или пробелов.

В Ansible есть две группы по умолчанию:

all	Все узлы в описи
ungrouped	Все узлы, которые не входят ни в одну другую группу (кроме all)

Группы также можно задавать на основе определенных шаблонов. Например, следующая группа включает диапазон имен, которые начинаются с `ans-web` и заканчиваются числом 1 или 2:

```
[webserver]
ans-web[1:2]
```

Шаблоны полезны, когда нужно управлять большим количеством узлов. Например, следующий шаблон охватывает все узлы в указанном диапазоне IP-адресов:

```
[all_servers]
172.16.191.[11:15]
```

Диапазоны определяются в формате `[начало:конец]` и включают все значения от *начала* до *конца* включительно, например: `[1:10]`, `[01:10]` и `[a:g]`.

Чтобы сделать одни группы вложенными в другие, используется суффикс `:children`. Например, гипотетически можно определить группу `[platforms]`, в которую вложены группы `[ubuntu]` и `[debian]`:

```
[platforms:children]
ubuntu
debian
```

Вернемся к файлу описи `hosts`, который мы создали ранее. После того как мы его сохранили, его синтаксис можно проверить с помощью такой команды:

```
ansible-inventory -i ./hosts --list --yaml
```

Вот что означают ключи этой команды:

<code>-i (--inventory)</code>	Файл описи (в нашем случае <code>./hosts</code>)
<code>--list</code>	Вывести текущую опись...
<code>--yaml</code>	...в формате YAML

Если проверка описи прошла успешно, команда выведет эквивалентный код YAML (по умолчанию `ansible-inventory` выводит данные в формате JSON):

```
packt@neptune:~/ansible$ ansible-inventory -i ./hosts --list --yaml
all:
  children:
    databases:
```

```
hosts:
  ans-db1: {}
  ans-db2: {}
webservers:
  hosts:
    ans-web1: {}
    ans-web2: {}
```

Мы составили файл описи Ansible в формате INI, но вместо этого могли бы использовать и YAML. Правда, из-за строгих требований к отступам и форматированию YAML может оказаться неудобным, особенно для крупных конфигураций. До конца этой главы мы продолжим использовать описи в формате INI.

Теперь давайте укажем Ansible на нашу опись. Добавьте в конфигурационный файл `./ansible.cfg` такие строки:

```
[defaults]
inventory = ~/ansible/hosts
```

После того как вы сохраните этот файл, все будет готово к тому, чтобы запускать команды или задачи Ansible на управляемых узлах. Управлять конфигурацией Ansible можно двумя способами: с помощью **разовых команд** и с помощью **сценариев Ansible**. Для начала рассмотрим разовые команды.

Разовые команды Ansible

Каждая разовая (*ad hoc*) команда выполняет одно задание Ansible и служит быстрым способом взаимодействия с управляемыми узлами. Эти простые операции полезны, если нужно внести небольшие изменения или протестировать систему.

Общий синтаксис разовой команды Ansible выглядит так:

```
ansible [ключи] шаблон -m модуль -a аргументы
```

Эта команда использует определенный *модуль* Ansible, чтобы выполнить ту или иную задачу на узлах, которые выбираются на основе *шаблона*. Задача описывается с помощью *аргументов*. Вы наверняка помните, что каждый модуль инкапсулирует определенную функциональность: например, отдельные модули позволяют управлять пользователями, пакетами и службами. Чтобы продемонстрировать, как используются разовые команды, мы задействуем некоторые из самых распространенных модулей Ansible. Начнем с модуля `ping`.

Модуль `ping`

Одна из самых простых разовых команд Ansible — `ping`, которая опрашивает управляемые узлы, чтобы проверить связь с ними по SSH и убедиться, что на них есть нужный интерпретатор Python. С шаблоном `all` она выводит сведения обо всех узлах:

```
packt@neptune:~/ansible$ ansible all -m ping
ans-db1 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python"
  },
  "changed": false,
  "ping": "pong"
}
# ... аналогичные блоки для узлов ans-db2, ans-web1 и ans-web2 ...
```

Вывод показывает, что команда была выполнена успешно (SUCCESS) и что на запрос ping удаленные узлы ответили "pong". Обратите внимание, что модуль ping в Ansible проверяет удаленное соединение с управляемыми узлами не через протокол ICMP. Это просто тестовый модуль, который работает с помощью Python; он не использует популярную команду оболочки ping, которая помогает устранять неполадки в сети.

Далее мы рассмотрим разовые команды с модулем user.

Модуль user

Вот еще один пример разовой команды, которая проверяет, существует ли определенный пользователь (packt) на всех узлах:

```
packt@neptune:~/ansible$ ansible all -m user -a "name=packt state=present"1
ans-web1 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  „append": false,
  „changed": false,
  „comment": „packt",
  „group": 1000,
  „home": „/home/packt",
  „move_home": false,
  „name": „packt",
  „shell": „/bin/bash",
  „state": „present",
  „uid": 1000
}
# ... аналогичные блоки для узлов ans-db2, ans-web1 и ans-web2 ...
```

Можно уточнить запрос, чтобы убедиться, что учетной записи packt соответствуют определенные идентификаторы пользователя и группы:

```
ansible all -m user -a "name=packt state=present uid=1000 group=1000"
```

¹ Параметр state по умолчанию принимает значение present, поэтому его можно специально не указывать. — *Примеч. ред.*

Шаблоны и модули Ansible

Разовые команды можно адресовать ограниченному подмножеству активов. Например, следующая команда опрашивает только узел `ans-web1`:

```
ansible ans-web1 -m ping
```

В шаблонах узлов можно использовать подстановочные знаки или имена групп, например:

```
ansible -m ping ans-web*
ansible -m ping webserver
```

Чтобы получить список всех модулей, которые доступны в Ansible, запустите такую команду:

```
ansible-doc --list
```

Чтобы найти определенный модуль, вывод этой команды можно перенаправить в `grep`. Если вас интересует более подробная информация о конкретном модуле, передайте его имя в качестве аргумента, например:

```
ansible-doc user
```

Просматривая вывод этой команды, обязательно ознакомьтесь с разделом **EXAMPLES**, в котором вы найдете практические примеры того, как использовать модуль в разовых командах и сценариях.

Как создавать новых пользователей

Модуль `user` также позволяет создавать новые учетные записи, если они нужны для тех или иных задач на управляемых узлах Ansible. Например, чтобы создать нового пользователя `webuser` на всех наших веб-серверах, можно запустить такую разовую команду:

```
ansible -bK webserver -m user -a "name=webuser state=present"
```

Поясним аргументы этой команды:

<code>-b</code> <code>--become¹</code>	Изменить контекст выполнения на <code>sudo</code> (<code>root</code>)
<code>-K</code> <code>--ask-become-pass</code>	Запрашивать пароль <code>sudo</code> на управляемых узлах; мы используем на них один и тот же пароль

¹ `become` (стать (кем-л. или чем-л.), превратиться (в кого-л. или во что-л.)) — функциональность Ansible, которая позволяет запускать на управляемых узлах команды от имени суперпользователя или других учетных записей, отличных от текущей. Подробнее см. docs.ansible.com/ansible/latest/playbook_guide/playbooks_privilege_escalation.html. — Примеч. ред.

<code>-m</code> <code>--module-name</code>	Использовать модуль <code>user</code>
<code>-a</code> <code>--args</code>	Аргументы модуля <code>user</code> в виде пар <i>ключ=значение</i> : <code>name=webuser</code> — имя пользователя <code>state=present</code> — создать новую учетную запись, только если записи с таким именем раньше не было
<code>webservers</code>	Группа управляемых узлов, которым адресована операция

Чтобы создать учетную запись на удаленном узле, нужно обладать административным доступом (`sudo`). Ключ `-b` (`--become`) **повышает привилегии** для команды Ansible, чтобы она выполнялась от имени суперпользователя.

По умолчанию Ansible не повышает привилегии **автоматически**. Если задача требует прав `sudo`, для нее нужно установить флаг `-b` (`--become`). Это поведение можно изменить в конфигурационном файле Ansible.

Чтобы Ansible автоматически повышал привилегии, добавьте в файл `ansible.cfg` следующие строки. После этого в разовых командах больше не понадобится указывать ключ `--b` (`--become`):

```
[privilege_escalation]
become = True
```

Если для суперпользователей на управляемых узлах включен вход по паролю, его понадобится задать, чтобы запустить разовую команду. Здесь пригодится ключ `-K` (`--ask-become-pass`), который заставляет Ansible запросить пароль со следующим сообщением:

```
BECOME password:
```

Этот пароль будет использоваться на всех управляемых узлах, которым адресована команда. Повышение привилегий связано с определенными проблемами безопасности, и в Ansible есть средства для того, чтобы снизить соответствующие риски. Подробнее об этом можно узнать на странице https://docs.ansible.com/ansible/latest/user_guide/become.html.

Как вы помните, мы отключили вход по паролю для суперпользователей на наших управляемых узлах (см. раздел «Учебная среда для Ansible»). Поэтому предыдущую разовую команду можно переписать так, чтобы Ansible не запрашивал повышения привилегий и соответствующего пароля:

```
packt@neptune:~/ansible$ ansible webservers -m user -a "name=webuser
state=present"
ans-web1 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },

```

```

...
„group”: 1001,
„home”: „/home/webuser”,
„move_home”: false,
„name”: „webuser”,
...
„uid”: 1001
}
# ... аналогичный блок для узла ans-web2 ...

```

Возможно, вы заметили, что часть вывода выделена, как и в предыдущих разовых командах. Ansible выделяет вывод, если он сообщает о том, что изменилось в *желаемом состоянии* управляемого узла. Если запустить ту же команду во второй раз, вывод не будет выделен: новых изменений не произойдет, потому что эта учетная запись уже создана. Это пример того, как работает *идемпотентная операция* Ansible.

Предыдущая команда создавала пользователя без пароля. В следующем разделе мы поговорим о том, как добавить или изменить пароль.

Как добавить или изменить пароль

Если вы изучите документацию по модулю `user` с помощью `ansible-doc user`, то узнаете, что в аргументах модуля можно использовать поле для пароля, но в этом случае Ansible принимает только **хеш-суммы паролей**. Чтобы захешировать пароль, мы задействуем `passlib` — вспомогательный модуль Python, который можно установить на управляющий узел Ansible с помощью такой команды:

```
pip install passlib
```

Обратите внимание, что здесь фигурирует система управления пакетами Python (`pip`). Если вы до этого установили Ansible с помощью `pip`, все будет в порядке, иначе вам стоит вернуться к разделу «Как установить Ansible с помощью `pip`».

После того как мы установили `passlib`, можно создать или изменить пароль пользователя с помощью такой команды:

```

ansible webserver -m user \
  -e "user_password=changeit!" \
  -a "name=webuser \
     update_password=always \
     password={{ user_password | password_hash('sha512') }}"

```

Вот параметры, которые относятся к паролю:

- `-e (--extra-vars)`: задает пользовательские переменные в формате *ключ=значение*. Здесь мы создали переменную `user_password` со значением `changeit!`, которое будет использоваться в параметре `password` в последней строке.
- `update_password=always`: обновляет пароль, если он отличается от предыдущего.

- `password={{ ... }}`: задает хеш-сумму пароля как значение выражения в двойных скобках.
- `user_password | password_hash('sha512')`: передает значение `changeit!` переменной `password` в функцию `password_hash()`, которая генерирует хеш-сумму в формате SHA-512. `password_hash()` входит в модуль `passlib`, который мы установили ранее.

Эта команда задает для пользователя `webuser` пароль `changeit!` и демонстрирует, как можно использовать переменные (в данном случае `password`) в разовых командах. В выводе команды вы увидите такую новую строку:

```
"password": "NOT_LOGGING_PASSWORD",
```

(По очевидным соображениям безопасности Ansible не показывает фактический пароль.)

Если вы теперь попытаетесь подключиться по SSH к любому из веб-серверов (`ans-web1` или `ans-web2`) с учетной записью `webuser`, вы должны успешно пройти аутентификацию с помощью пароля `changeit!`.

Как удалить пользователя

Чтобы удалить учетную запись `webuser` на всех веб-серверах, запустите такую разовую команду:

```
ansible webserver -m user -a "name=webuser state=absent remove=yes force=yes"
```

Параметр `state=absent` удаляет учетную запись `webuser`. Параметры `remove` и `force` эквивалентны команде `userdel -rf`: они удаляют домашний каталог пользователя и все файлы в нем, даже если они не принадлежат пользователю.

Вывод команды выглядит так:

```
ans-web1 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": true,
  "force": true,
  "name": "webuser",
  "remove": true,
  "state": "absent",
  "stderr": "userdel: webuser mail spool (/var/mail/webuser) not found\n",
  "stderr_lines": [
    "userdel: webuser mail spool (/var/mail/webuser) not found"
  ]
}
# ... аналогичный блок для узла ans-web2 ...
```

Можете не обращать внимания на параметры `stderr` и `stderr_lines`: на самом деле здесь нет никаких ошибок, потому что пользователь ранее не создавал почтовый буфер.

Далее мы рассмотрим модуль `package` и запустим с ним несколько разовых команд.

Модуль *package*

Модуль `package` позволяет управлять пакетами программного обеспечения. Например, следующая команда устанавливает веб-сервер **nginx** на все узлы в группе `webservers`:

```
packt@neptune:~/ansible$ ansible webservers -m package -a "name=nginx
state=present"
ans-web1 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "cache_update_time": 1696346347,
  "cache_updated": false,
  "changed": true,
  "stderr": "",
  "stderr_lines": [],
  "stdout": "Reading package lists...\nBuilding dependency tree...\n ..."
}
```

С помощью похожей разовой команды можно установить сервер баз данных **MySQL** на все узлы в группе `databases`:

```
ansible databases -m package -a "name=mysql-server state=present"
```

Если нужно удалить пакет, запустите такую же разовую команду, но укажите `state=absent` вместо `state=present`.

Хотя модуль `package` обеспечивает хорошую кроссплатформенную абстракцию на уровне операционной системы, некоторые операции с пакетами лучше выполнять через системы управления пакетами для конкретной платформы. Далее мы покажем, как использовать модуль `apt`.

Модули систем управления пакетами для конкретных платформ

Следующая разовая команда устанавливает последние обновления на все управляемые узлы в группе `webservers`:

```
ansible webservers -m apt -a "upgrade=dist update_cache=yes"
```

Модули управления пакетами для конкретных платформ (`apt`, `yum` и т. д.) делают все то же самое, что и модуль `package`, который не зависит от платформы, и обеспечивают дополнительные функции, которые специфичны для той или иной системы.

Далее мы рассмотрим модуль `service` и запустим с ним несколько разовых команд.

Модуль *service*

Следующая команда перезапускает службу `nginx` на всех узлах в группе `webserver`s:

```
packt@neptune:~/ansible$ ansible webserver -m service -a "name=nginx
state=restarted"
ans-web1 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3"
  },
  "changed": true,
  "name": "nginx" ,
  "state": "started",
  "status": {
    "ActiveEnterTimestamp": "Tue 2023-10-03 18:00:31 UTC",
    "ActiveEnterTimestampMonotonic": "8394578958",
    "ActiveExitTimestamp": "n/a",
    "ActiveExitTimestampMonotonic": "0" ,
    "ActiveState": "active",
    ...
  }
}
```

Похожим образом можно перезапустить службу `mysql` на всех серверах баз данных, но здесь есть своя хитрость. В одних системах (например, Ubuntu) служба MySQL называется `mysql`, а в других (например, Debian) — `mysqld`. Конечно, мы могли бы указать нужное имя службы для каждого отдельного узла, но если у нас много серверов, это стало бы весьма трудоемкой задачей. Вместо этого можно использовать *шаблон исключения* (обозначается знаком `!`), чтобы исключить определенные узлы или группы.

Предположим, что у нас есть группы `ubuntu` и `debian`, в каждую из которых входят все узлы под управлением одноименных операционных систем. Следующая команда перезапустит службу `mysql` на всех узлах в группе `databases`, кроме тех, которые входят в группу `debian`:

```
ansible 'databases:!debian' -m service -a "name=mysql state=restarted"
```

Аналогичным образом можно перезапустить службу `mysqld` на всех узлах в группе `databases`, за исключением тех, которые входят в группу `ubuntu`:

```
ansible 'databases:!ubuntu' -m service -a "name=mysqld state=restarted"
```

Чтобы применить шаблон исключения, используйте одинарные кавычки (`'...'`), иначе команда `ansible` не выполнится.

Давайте рассмотрим последний модуль Ansible и связанную с ним разовую команду, которую часто используют, если нужно обновить систему.

Модуль *reboot*

Следующая разовая команда перезагружает все узлы в группе `webserver`s:

```
ansible webserver -m reboot -a "reboot_timeout=3600"
```

Чтобы перезагрузиться, узлам может потребоваться значительное время, особенно при крупных обновлениях, поэтому мы увеличили время ожидания перезагрузки до 3600 секунд по сравнению с 600 секундами по умолчанию.

В нашем случае перезагрузка заняла всего несколько секунд, и команда вывела такие данные:

```
ans-web1 | CHANGED => {
  "changed": true,
  "elapsed": 10,
  "rebooted": true
}
ans-web2 | CHANGED => {
  "changed": true,
  "elapsed": 11,
  "rebooted": true
}
```

Этот пример завершает перечень разовых команд, в которых используются различные модули. В следующем разделе мы представим краткий обзор распространенных модулей Ansible и расскажем, как познакомиться с ними подробнее.

Обзор модулей Ansible

Ansible предлагает обширную библиотеку модулей. Как мы уже отмечали, команда `ansible-doc -list` выводит список всех доступных модулей. С ним также можно ознакомиться на соответствующей странице официальной документации: docs.ansible.com/ansible/latest/collections/index_module.html.

Вот некоторые модули, которые особенно широко используются в повседневных задачах системного администрирования и управления конфигурацией с помощью Ansible:

Модули управления пакетами	
apt	Управляет пакетами APT
dnf	Управляет пакетами DNF и YUM
Системные модули	
user	Управляет пользователями
service	Управляет службами
reboot	Перезагружает узлы
firewalld	Управляет межсетевыми экранами
Файловые модули	
copy	Позволяет копировать файлы на управляемые узлы
synchronize	Синхронизирует файлы и каталоги с помощью rsync
file	Управляет правами доступа и атрибутами файлов
lineinfile	Манипулирует строками в текстовых файлах

Сетевые модули	
<code>nmcli</code>	Управляет сетевой конфигурацией
<code>get_url</code>	Позволяет загружать файлы на узлы по протоколам HTTP, HTTPS и FTP
<code>uri</code>	Позволяет взаимодействовать с веб-службами и оконечными точками API
Модули команд (не идемпотентные)	
<code>raw</code>	Позволяет небезопасно запускать команды на удаленном узле через SSH; не требует, чтобы на узле был установлен Python
<code>command</code>	Позволяет безопасно запускать команды на удаленном узле с помощью контекста удаленного выполнения Python
<code>shell</code>	Позволяет запускать команды оболочки на управляемых узлах

Отметим, что разовые команды всегда выполняют *одну операцию* с помощью *одного модуля*. Это дает свои преимущества (например, когда нужно быстро внести простые изменения), но накладывает и определенные ограничения. Для более сложных задач управления конфигурацией используются сценарии Ansible. В следующем разделе мы узнаем, как их создавать и запускать.

Как работать со сценариями Ansible

Сценарий (плейбук¹) Ansible — это, по сути, список задач, которые выполняются автоматически. Управление конфигурацией в Ansible в основном происходит с помощью сценариев. Сценарий — это файл в формате YAML, который состоит из одной или нескольких **групп задач** (`play`), а каждая группа содержит список **задач**, которые выполняются в том порядке, в каком они перечислены. Сценарий — это единица выполнения, которая запускает задачи на наборе узлов, а узлы выбираются по идентификатору группы или по шаблону. В каждой задаче используется один модуль, который выполняет определенное действие, направленное на удаленный узел. Отдельную задачу можно рассматривать как простую разовую команду Ansible. Поскольку большинство модулей Ansible соответствуют идемпотентным контекстам выполнения, сценарии тоже идемпотентны: многократный запуск одного и того же сценария приводит к тому же результату, что и однократный.

Хорошо написанные сценарии Ansible помогают заменить трудоемкие административные задачи и сложные сценарии командной оболочки относительно простыми и удобными в сопровождении манифестами, которые выполняют легко воспроизводимые и предсказуемые процедуры.

Далее мы создадим наш первый сценарий Ansible.

¹ Ansible позаимствовал основные термины из спорта: например, `playbook` — это руководство по стратегиям и тактикам, которые команда применяет в игре; `play` — это отдельная комбинация, которую команда разыгрывает на поле. — *Примеч. пер.*

Как создать простой сценарий

Наш первый сценарий будет опираться на разовую команду, с помощью которой мы ранее создали пользователя (**webuser**). Напомним, что она выглядела так:

```
ansible webservers -m user -a "name=webuser state=present"
```

Когда наш сценарий будет готов, вы заметите, что в нем много общего с этой командой.

Редактируя сценарий, не забывайте о правилах форматирования YAML:

- Делайте отступы только пробелами (без табуляций).
- Делайте отступы единообразной длины (например, по два пробела на каждый уровень).
- Задавайте одинаковый отступ элементам одного иерархического уровня.
- Делайте отступы дочерних элементов на один шаг больше, чем у родительского элемента.

Теперь давайте создадим файл сценария **create-user.yml** в том же каталоге проекта **~/ansible**, где уже находится наша текущая опись (**hosts**) и конфигурационный файл Ansible **ansible.cfg**. В текстовом редакторе добавьте в файл такие строки:

```
---
- name: Create a specific user on all web servers
  # Создать одинаковые учетные записи на всех веб-серверах
  hosts: webservers
  become: yes
  tasks:
    - name: Create the 'webuser' account
      # Создать учетную запись webuser
      user:
        name: webuser
        state: present
```

Давайте рассмотрим каждую строку в этом сценарии:

---	Начало сценария
- name	Название блока (например, сценария или группы задач)
hosts: webservers	Сценарий адресован узлам в группе webservers

¹ В Ansible 2.10 и более поздних версиях рекомендуется использовать полные имена модулей и других коллекций: например, **ansible.builtin.user**, а не просто **user**. — *Примеч. ред.*

<code>become: yes</code>	Повышать привилегии для текущей задачи. Эту строку можно не указывать, если вы включили неуправляемое повышение привилегий в конфигурационном файле Ansible (<code>become = True</code> в разделе <code>[privileged_escalation]</code>)
<code>tasks</code>	Список задач в текущей группе
<code>user</code>	Модуль, который использует текущая задача ¹
<code>name</code>	Имя учетной записи, которую создает сценарий
<code>state: present</code>	Желаемое состояние управляемого узла: мы хотим, чтобы учетная запись присутствовала в системе

Давайте запустим наш сценарий:

```
packt@neptune:~/ansible$ ansible-playbook create-user.yml
```

```
PLAY [Create a specific user on all web servers] *****
```

```
TASK [Gathering Facts] *****
```

```
ok: [ans-web1]
```

```
ok: [ans-web2]
```

```
TASK [Create the 'webuser' account] *****
```

```
changed: [ans-web1]
```

```
changed: [ans-web2]
```

```
PLAY RECAP *****
```

```
ans-web1  : ok=2 changed=1 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0
```

```
ans-web2  : ok=2 changed=1 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0
```

Большинство ключей команды `ansible-playbook` аналогичны ключам `ansible`. Вот некоторые из них:

<code>-i</code>	<code>--inventory</code>	Путь к файлу описи
<code>-b</code>	<code>--become</code>	Повышает привилегии до <code>sudo (root)</code>
<code>-C</code>	<code>--check</code>	Запускает сценарий в пробном режиме и выводит потенциальный результат, не внося реальных изменений в систему. Этот ключ позволяет проверить сценарий перед тем, как запускать
<code>-l</code>	<code>--limit</code>	Ограничивает действие команды или сценария подмножеством управляемых узлов
<code>--syntax-check</code>		Проверяет синтаксис сценария, не внося изменений в систему; доступен только для команды <code>ansible-playbook</code>

Давайте в качестве упражнения напишем еще один сценарий, который на этот раз удаляет учетную запись. Назовите файл сценария `delete-user.yml` и добавьте в него такое содержимое:

```

---
- name: Delete a specific user on all web servers
  # Удалить одноименные учетные записи на всех веб-серверах
  hosts: webservers
  become: yes
  tasks:
    - name: Delete the 'webuser' account
      # Удалить учетную запись webuser
      user:
        name: webuser
        remove: yes
        force: yes
        state: absent

```

Запустите этот сценарий так же, как предыдущий, и убедитесь, что он выводит примерно такие же результаты:

```
ansible-playbook delete-user.yml
```

Дальше мы поговорим о том, как можно еще эффективнее оптимизировать рабочие процессы управления конфигурацией. Прежде всего мы рассмотрим переменные в сценариях.

Как использовать переменные в сценариях Ansible

Ansible предоставляет гибкую и универсальную модель для работы с переменными как в сценариях, так и в специальных командах. С помощью переменных мы фактически *параметризуем* сценарий, отчего он становится динамическим и пригодным для повторного использования.

Возьмем предыдущий сценарий, который создавал учетную запись. В нем мы жестко закодировали имя пользователя (**webuser**), поэтому этот сценарий нельзя повторно использовать, чтобы создать другую учетную запись (например, **webadmin**), если только не добавить в него соответствующую задачу. Но тогда чем больше будет пользователей, тем больше разрастется наш сценарий и тем сложнее будет его сопровождать. А что, если мы захотим задать пароль для каждого пользователя? Тогда сценарий станет еще сложнее.

Именно здесь в дело вступают **переменные**. Давайте сначала разберемся, что это такое и как они записываются.

Как устроены переменные в Ansible

Вместо жестко заданных значений в сценарии можно использовать переменные, и тогда он станет динамичным. Сценарий, который создает учетную запись с определенным именем и паролем, на псевдокоде будет запускаться так:

```
Пользователь = Сценарий(имя_пользователя, пароль)
```

Переменные в Ansible заключаются в двойные фигурные скобки, например: `{{ username }}`. Давайте посмотрим, как их можно использовать в сценариях.

Измените три строки в файле `create-user.yml`, над которым мы работали в предыдущем разделе:

```
---
- name: Create a specific user on all web servers
  # Создать одинаковые учетные записи на всех веб-серверах
  hosts: webserver
  become: yes
  tasks:
    - name: Create the '{{ username }}' account
      # Создать учетную запись с именем из значения переменной username
      user:
        name: "{{ username }}"
        state: present
```

Теперь вместо жестко закодированного значения `webuser` мы используем переменную `{{ username }}`. Мы окружили ее кавычками, чтобы код не конфликтовал с нотацией словаря YAML. Имена переменных в Ansible должны начинаться с буквы и могут содержать только алфавитно-цифровые символы и знаки подчеркивания.

Как задавать значения переменных

Давайте разберемся, *как* и *где* задавать значения переменных. Для этого в Ansible реализована иерархическая модель:

1. **Глобальные переменные.** Значения задаются для всех узлов либо через ключ `--extra-vars` команды `ansible-playbook`, либо в файле `./group_vars/all`.
2. **Переменные группы узлов.** Значения задаются для узлов определенной группы либо в файле описи, либо в файле, который называется так же, как группа, и находится в локальном каталоге `./group_vars`.
3. **Переменные узла.** Значения задаются для конкретного узла либо в файле описи, либо в файле, который называется так же, как узел, и находится в локальном каталоге `./host_vars`. Переменные для конкретного узла также доступны из так называемых **фактов Ansible**, о которых можно подробнее узнать на странице docs.ansible.com/ansible/latest/user_guide/playbooks_vars_facts.html.
4. **Переменные группы задач.** Значения задаются в контексте текущей группы задач для узлов, на которые действует эта группа; например, с помощью директивы `vars` в группе задач или модуля `include_vars`.

Пункты этого списка расположены от меньшего приоритета к большему. Другими словами, если значение переменной задано в группе задач, то его приоритет будет выше, чем у такой же переменной, которая задана на уровне узла, группы или на глобальном уровне.

В качестве примера можно вспомнить, что на платформах Ubuntu служба MySQL называется `mysql`, а в RHEL/Fedora — `mysqld`. Допустим, мы хотим перезапустить службу MySQL на всех узлах в группе `databases`. Если предположить, что большинство наших серверов баз данных работают под управлением Ubuntu, мы

можем определить переменную `service` на уровне группы узлов как `service:mysql` и задать ее в локальном файле проекта `./group_vars/databases`. Затем на уровне группы задач, которые управляют состоянием службы, можно переопределить значение переменной `service` как `mysqld` в тех случаях, если удаленный узел работает под управлением Fedora.

Давайте на практических примерах закрепим свои знания о том, где размещать переменные и как задавать их значения. Вернувшись в сценарий `create-user.yml`, можно определить переменную `username` на уровне группы задач с помощью такой директивы:

```
...
become: yes
vars:
  username: webuser
tasks:
...
```

Запустите сценарий с помощью команды `ansible-playbook` и посмотрите на результат.

Аналогичным образом можно добавить переменную `username` в файл сценария `delete-user.yml`, который удаляет учетные записи. Выполните это упражнение самостоятельно и убедитесь, что сценарий с переменными работает так же, как с явными значениями.

Как усовершенствовать сценарии

Давайте посмотрим, как еще можно улучшить наши сценарии `create-user` и `delete-user`. Например, можно сделать так:

1. Поскольку группа задач нацелена исключительно на группу `webservers`, переменную `username` можно определить в файле `./group_vars/webservers`, чтобы сценарии стали компактнее. Удалите определения этой переменной из обоих файлов.
2. Затем создайте в локальном каталоге `~/ansible` подкаталог `group_vars`, а в нем — файл `webservers.yml` с таким содержанием:

```
username: webuser
```

Этот файл можно было назвать и просто `webservers`, чтобы его имя совпадало с именем группы узлов. Однако мы предпочли добавить расширение `.yml`, чтобы подчеркнуть, что файл составлен в формате YAML. Ansible распознает и то и другое имя. Вот как сейчас выглядит структура каталогов нашего проекта:

```
packt@neptune:~/ansible$ tree
.
├── ansible.cfg
├── create-user.yml
└── delete-user.yml
```

```

├── group_vars
│   └── webservers.yml
├── hosts
└── test.cfg

```

Если запустить обновленные сценарии, результаты должны быть такими же, как в прошлый раз:

```

ansible-playbook create-user.yml
ansible-playbook delete-user.yml

```

3. Теперь давайте добавим в сценарий `create-user` еще одну переменную — `password` (пароль пользователя). Вспомните разовую команду, которую мы раньше использовали для этого. Если нужно, обратитесь к разделу «Специальные команды Ansible» ранее в этой главе.

Добавьте в задачу в файле `create-user.yml` такие строки:

```

...
user:
  name: webuser
  name: "{{ username }}"
  password: „{{ password | password_hash(,sha512') }}"
  update_password: always
  state: present

```

Как вы могли заметить, эти строки похожи на аргументы соответствующей разовой команды.

4. Теперь добавьте в файл `./group_vars/webservers.yml` переменную `password` со значением `changeit!`:

```

username: webuser
password: changeit!

```

5. Снова запустите сценарий и убедитесь, что он приводит к прежнему результату:

```
ansible-playbook create-user.yml
```

6. Чтобы проверить, что на веб-сервере (например, `ans-web1`) действительно создана новая учетная запись `webuser` с паролем `changeit!`, попробуйте подключиться к этому серверу по SSH:

```
ssh webuser@ans-web1
```

Когда вы введете пароль `changeit!`, вы должны успешно войти в систему. Выйдите из удаленного терминала обратно, прежде чем приступать к следующим шагам.

7. Чтобы вернуться к исходному состоянию, давайте удалим учетную запись `webuser` на веб-серверах:

```
ansible-playbook delete-user.yml
```

8. Теперь предположим, что мы хотели бы повторно использовать сценарий `create-user`, чтобы создать другого пользователя (допустим, `webadmin`) с дру-

гим паролем (допустим, `changeme!`). Один из способов решить эту задачу — использовать ключ `-e (--extra-vars)` в команде `ansible-playbook`:

```
ansible-playbook -e '{"username": "webadmin", "password": "changeme!"}'  
create-user.yml
```

С помощью следующей команды можно убедиться, что на управляемом узле теперь есть учетная запись `webadmin` с соответствующим паролем:

```
ssh webadmin@ans-web1
```

Как и в предыдущем упражнении, выйдите из удаленного терминала, прежде чем продолжать.

Как видите, ключ `-e (--extra-vars)` принимает строку в формате JSON, которая содержит имена и значения полей `username` и `password`. Эти значения будут *переопределять* одноименные переменные, которые объявлялись на уровне группы узлов в файле `./group_vars/webserver.yml`.

9. Прежде чем переходить к следующим упражнениям, давайте удалим учетные записи `webuser` и `webadmin`. Чтобы удалить `webuser`, запустите сценарий `delete-user` без параметров:

```
ansible-playbook delete-user.yml
```

10. Теперь с помощью ключа `-e` удалите учетную запись `webadmin`:

```
ansible-playbook -e '{"username": "webadmin"}' delete-user.yml
```

С помощью ключа `--extra-vars` можно запускать сценарии `create-user` и `delete-user` так, чтобы они работали с различными учетными записями. Сценарии можно запускать вручную или в цикле, передавая данные JSON с необходимыми значениями. Хотя все это можно реализовать с помощью сценария командной оболочки, Ansible позволяет еще больше оптимизировать сценарии, если итерировать задачи с помощью циклов. Мы поговорим о циклах позже в этой главе, но сначала давайте защитим наши пароли с помощью механизмов Ansible, которые позволяют управлять секретами.

Как работать с секретами

В Ansible есть специальный модуль **Ansible vault**, который позволяет управлять секретами. С его помощью можно шифровать и хранить конфиденциальные данные, такие как переменные или файлы, на которые ссылаются сценарии. По сути, Ansible vault — это хранилище данных в формате «ключ — значение», которое защищено паролем. Чтобы управлять секретами, можно использовать утилиту командной строки `ansible-vault`. Для сценария, который создает учетную запись с паролем, имеет смысл сделать так, чтобы пароль не хранился в открытом виде. Напомним, что сейчас пароль содержится в файле `./group_vars/webserver.yml`:

```
---  
username: webuser  
password: changeit!
```

Последняя строка содержит конфиденциальные данные: пароль отображается в виде обычного текста. Есть несколько способов защитить эти данные:

- Зашифровать файл `webservers.yml`. Если мы пойдем по этому пути, то вместе с паролем придется шифровать несекретные данные — например, имя пользователя или другую информацию общего назначения. Если у нас много пользователей, это приведет к избыточным накладным расходам на то, чтобы шифровать и дешифровать лишние данные.
- Зашифровать только переменную `password`. Это отлично подойдет для одного пользователя, но если их будет много, то нам придется иметь дело с несколькими переменными паролей, каждую из которых надо шифровать и дешифровать по-своему. Если у нас много пользователей, это опять приведет к проблемам с производительностью.
- Хранить пароль в отдельном защищенном файле. В идеале у нас должен быть отдельный файл, в котором хранятся все конфиденциальные данные и который дешифруется только один раз, когда запускается сценарий, даже если в нем содержится несколько паролей.

Мы пойдем по третьему пути и создадим отдельный файл, в котором будут храниться пароли пользователей.

Как защитить конфиденциальные данные

Вот что нужно сделать, чтобы обеспечить безопасность наших данных:

1. Создайте в каталоге `~/ansible/` файл `passwords.yml` и добавьте в него такое содержимое:

```
---
webuser:
  password: changeit!
```

2. Мы добавили элемент словаря YAML, ключ которого — имя `webuser`, а значение — другой элемент словаря в виде пары «ключ — значение»: `password: changeit!`. Эквивалентное представление YAML выглядит так:

```
webuser: { password: changeit! }
```

В таком же режиме можно добавлять пароли для разных пользователей, например, так:

```
webuser: { password: changeit! }
webadmin: { password: changeme! }
```

Мы подробнее рассмотрим эту структуру данных и узнаем, как ее использовать, далее в этом разделе, когда будем применять переменную `password` в сценарии.

3. Теперь, поскольку пароль хранится в другом файле, мы удалим соответствующую строку из `webservers.yml`. Давайте также для примера добавим другую

информацию о пользователе с помощью параметра `comment`. Теперь файл `webusers.yml` выглядит так:

```
---
username: webuser
comment: Обычный пользователь
```

4. Чтобы защитить наши данные, зашифруйте файл `passwords.yml` с помощью Ansible vault:

```
ansible-vault encrypt passwords.yml
```

Команда предложит вам создать пароль хранилища для защиты файла¹. Придумайте и запомните пароль, потому что мы будем использовать его на протяжении всего этого раздела.

5. Просмотрите содержимое файла `passwords.yml` с помощью `cat` и убедитесь, что оно зашифровано:

```
packt@neptune:~/ansible$ cat passwords.yml
ANSIBLE_VAULT;1.1;AES256
6434373337303631616334336135383137623263346331346335366436303863623633062353
2643137313665636236663937323238303731363939366134390a356230663463656365353061
65646639383832363339616239666161383865653966623764626462333464306636666562303
8623937323132626563323536300a386261613761363835656233376139653030633831383235
32333238356664313930396330646230326664663238356264623065346563653261633835613
6623739663135663236663436376437636232663537353464366663323431643833
```

6. Подкоманда `view` позволяет дешифровать содержимое файла `passwords.yml`:

```
packt@neptune:~/ansible$ ansible-vault view passwords.yml
```

```
Vault password:
---
webuser:
  password: changeit!
```

Обратите внимание, что команда запрашивает пароль хранилища, который мы создали ранее. На выходе вы увидите код в формате YAML и сможете убедиться, что он совпадает с исходным содержанием файла `passwords.yml`.

7. Если нужно изменить зашифрованный файл, его можно отредактировать с помощью такой команды:

```
ansible-vault edit passwords.yml
```

После того как вы введете пароль хранилища, эта команда откроет локальный редактор (`vi`), в котором вы сможете внести изменения. Если вы хотите повторно зашифровать защищенный файл с другим паролем, запустите такую команду:

```
ansible-vault rekey passwords.yml
```

¹ Это не тот же пароль `changeit!`, который мы, собственно, защищаем в этом примере. — *Примеч. ред.*

Эта команда запросит текущий пароль хранилища, а затем предложит задать новый пароль.

Теперь давайте узнаем, как ссылаться на конфиденциальные данные (секреты) в сценарии Ansible.

Ссылки на секреты в сценариях Ansible

Обратиться к конфиденциальным данным можно так:

1. Сначала давайте убедимся, что мы можем прочесть пароль из хранилища. Создайте новый файл `create-user-new.yml` с таким содержанием:

```
---
- name: Create a specific user on all web servers
  hosts: webservers
  become: yes
  tasks:

  - name: Get the password for {{ username }} from Vault
    # Получить из хранилища пароль для учетной записи,
    # имя которой хранится в переменной username
    include_vars:
      file: passwords.yml

  - name: Output password for {{ username }}
    # Вывести пароль для учетной записи
    debug:
      msg: "{{ vars [username]['password'] }}"
```

Мы добавили две задачи:

- `include_vars` читает переменные из файла `passwords.yml`;
 - `debug` выводит отладочную информацию — в данном случае пароль, который сценарий считал из хранилища.
2. Ни одна из этих задач *не знает*, что файл `passwords.yml` зашифрован. Вся черная магия происходит в этой строке:

```
msg: "{{ vars [username]['password'] }}"
```

Здесь мы использовали словарь `vars[ ]`, который позволяет обратиться к определенной переменной в сценарии. `vars[ ]` — это *зарезервированная* структура данных, в которой хранятся все переменные, созданные в сценарии с помощью `vars` и `include_vars`. В квадратных скобках мы указали ключ `username`, чтобы вся конструкция возвратила значение по этому ключу:

```
{{ vars[username] }}
```

Сценарий получает значение `username` из файла `./group_vars/webservers.yml`, и это значение — `webuser`. Следовательно, `vars[webuser]` возвращает соответствующую запись из файла `passwords.yml`:

```
webuser: { password: changeit! }
```

3. Чтобы теперь из этой записи получить значение пароля, мы добавляем к предыдущей конструкции ключ 'password':

```
{{ vars[username]['password'] }}
```

4. Запустите этот сценарий с помощью такой команды:

```
ansible-playbook --ask-vault-pass create-user-new.yml
```

Мы использовали ключ `--ask-vault-pass`, чтобы Ansible знал, что сценарию нужен доступ к хранилищу, иначе он завершится с ошибкой. Убедитесь, что в выводе этой команды отражены наши пароли, которые сценарий успешно извлек из хранилища:

```
TASK (Debug password for webuser) *****
ok: [ans-web1] => {
    "msg": "changeit!"
}
# ... аналогичная запись для узла ans-web2 ...
```

5. Давайте продолжим совершенствовать наш сценарий `create-user-new.yml` и изменим его содержимое таким образом:

```
---
- name: Create a specific user on all web servers
  # Создать одинаковые учетные записи на всех веб-серверах
  hosts: webserver
  become: yes

  vars:
    # 1
    password: "{{ vars[username]['password'] }}"

  tasks:
    # 2

    - name: Get the password for {{ username }} from Vault
      # Получить из хранилища пароль для учетной записи,
      # имя которой хранится в переменной username
      include_vars:
        file: passwords.yml

    - name: Output password for {{ username }} # 3
      # Вывести пароль для учетной записи
      debug:
        msg: "{{ password }}"
      no_log: true

    - name: Create the '{{ username }}' account # 4
      # Создать учетную запись
      user:
        name: "{{ username }}"
        comment: "{{ comment }}"
        password: "{{ password | password_hash('sha512') }}"
        update_password: always
        state: present
```

Поясним некоторые особенности этой реализации:

- Мы добавили блок `vars` (# 1), чтобы определить локальную переменную `password`, которая находится в области видимости группы команд и в которую считывается пароль из хранилища. Мы повторно используем переменную `password` в нескольких задачах.
- Задача `include_vars` (# 2) добавляет внешнюю ссылку на переменные, которые определены в защищенном файле `passwords.yml`.
- Задача `debug` (# 3) в предыдущем упражнении помогала убедиться, что сценарий считывает пароль из хранилища. Ее можно удалить, но можно и сохранить, чтобы использовать в дальнейшем. Если вы не удалите задачу, убедитесь, что у вас включен параметр `no_log: true`, чтобы конфиденциальная информация не выводилась в терминал. При отладке можно временно установить `no_log: false`.
- Задача `user` (# 4) считывает значение переменной `password` и хеширует его: это нужно модулю `user` в целях безопасности. Кроме того, мы добавили поле `comment`, которое сопоставляется с записью GECOS, чтобы вывести дополнительную информацию о пользователе. (См. раздел «Как создавать учетные записи пользователей с помощью `useradd`» в главе 4 «Как управлять пользователями и группами».)

6. Запустите сценарий с помощью уже знакомой команды:

```
ansible-playbook --ask-vault-pass create-user-new.yml
```

После того как команда выполнится, вы можете проверить новую учетную запись таким образом:

- Подключитесь по SSH к любому из веб-серверов, используя соответствующие имя пользователя и пароль, например:

```
ssh webuser@ans-web1
```

- На этом сервере найдите учетную запись `webuser` в файле `/etc/passwd`:

```
#!/ cat /etc/passwd | grep webuser
webuser:x:1001:1001:Обычный пользователь:/home/webuser:/bin/sh
```

Обратите внимание, что в выводе присутствует поле GECOS.

Часто бывает нужно запустить команду `ansible-playbook`, не вводя пароль хранилища, как того требует параметр `--ask-vault-pass`. Доступ без пароля нужен в сценариях командной оболочки или в автоматизированных рабочих процессах, где используется Ansible vault. Чтобы пароль хранилища стал автоматически доступен при запуске сценария, для начала запишите этот пароль в обычный текстовый файл (например, `vault.pass`), предпочтительно в домашнем каталоге. Теперь, чтобы использовать этот файл в сценариях Ansible, выберите *один из двух* вариантов (не оба сразу!):

- Создайте переменную окружения:

```
export ANSIBLE_VAULT_PASSWORD_FILE=~/.vault.pass
```

- Добавьте строку в раздел `[defaults]` файла `ansible.cfg`:

```
vault_password_file = ~/vault.pass
```

Теперь сценарий `create-user-new` можно запускать без ключа `--ask-vault-pass`:

```
ansible-playbook create-user-new.yml
```

Может показаться небезопасным, что несколько секретов защищены с помощью одного пароля хранилища. Ansible поддерживает множественные пароли хранилищ с помощью идентификаторов хранилищ.

Идентификаторы хранилищ

Идентификатор хранилища (vault ID) — это метка, которая связана с одним или несколькими секретами хранилища. Каждому идентификатору соответствует уникальный пароль, с помощью которого можно шифровать и дешифровать соответствующие секреты. Чтобы проиллюстрировать идентификаторы хранилища на практике, давайте предположим, что мы хотим с их помощью защитить наш файл `passwords.yml`. Следующая команда создает идентификатор `passwords` и предлагает задать пароль:

```
ansible-vault create --vault-id passwords@prompt passwords.yml
```

Теперь предположим, что мы также хотим защитить некоторые пользовательские ключи API, которые хранятся в файле `apikeys.yml`. Аналогичная команда создает соответствующий идентификатор хранилища с именем `apikeys`:

```
ansible-vault create --vault-id apikeys@prompt apikeys.yml
```

Итак, мы создали два идентификатора хранилища, каждый из которых заблокирован своим паролем и защищает разные ресурсы. Идентификаторы хранилища обеспечивают улучшенный контекст безопасности: если один из паролей идентификатора будет скомпрометирован, это не затронет ресурсы, которые защищены другими идентификаторами.

- Идентификаторы хранилища также позволяют настроить различные уровни доступа к секретам. В частности, можно задать разные идентификаторы для разных групп пользователей, например: `admin` (администраторы), `dev` (разработчики) и `test` (тестировщики). Кроме того, если вы сопровождаете несколько проектов по управлению конфигурацией, у каждого из них могут быть свои собственные идентификаторы хранилища и секреты: например, `user-config` (пользовательские системы), `web-config` (веб-серверы) и `db-config` (серверы баз данных).
- Один и тот же идентификатор хранилища можно связать с несколькими секретами. Например, следующая команда создает идентификатор хранилища `user-config`, который защищает файлы `passwords.yml` и `api-keys.yml`:

```
ansible-vault create --vault-id user-config@prompt passwords.yml apikeys.yml
```

- Команде `ansible-vault` также можно передать файл, в котором находится нужный пароль хранилища. Например, следующая команда шифрует файл `apikeys.yml`, который защищен паролем идентификатора из файла `apikeys.pass`:

```
ansible-vault encrypt --vault-id apikeys@apikeys.pass apikeys.yml
```

Файлы с паролями к хранилищам можно называть как угодно, но если придерживаться определенной системы именования (например, чтобы названия файлов совпадали с соответствующими идентификаторами хранилища), это облегчит вам работу, если вы управляете множественными секретами.

- Идентификатор хранилища (например, `passwords`) можно передать сценарию (например, `create-users-new.yml`) с помощью такой команды:

```
ansible-playbook --vault-id passwords@passwords.pass create-users-new.yml
```

Чтобы получить дополнительную информацию об Ansible vault, обратитесь к официальной документации: docs.ansible.com/ansible/latest/vault_guide/index.html.

До сих пор мы создавали учетную запись одного пользователя с паролем. А что, если нужно создать нескольких пользователей, у каждого из которых свой пароль? Как мы уже отмечали, теоретически можно запустить сценарий `create-user` и переопределить переменные `username` и `password` с помощью ключа `--extra-vars`. Но это не самый эффективный способ, не говоря уже о том, что такое решение сложно сопровождать. В следующем разделе мы рассмотрим, как повторять задачи в цикле в сценариях Ansible.

Циклы в сценариях Ansible

С помощью **циклов** можно эффективно выполнять повторяющиеся задачи в сценариях Ansible. В Ansible есть два основных типа циклов:

- `loop`: рекомендуемый способ итерации по коллекции.
- `with_коллекция`: реализации циклов, специфичные для отдельных коллекций, например `with_list`, `with_items` и `with_dict`.

В этом разделе мы сосредоточимся на конструкции `loop` (эквивалентной `with_list`), которая лучше всего подходит для простых циклов. Давайте расширим наш предыдущий сценарий так, чтобы он создавал несколько пользователей. Для начала сравним, как выполняются повторяющиеся задачи с циклами и без них:

1. Прежде всего, убедитесь, что `~/ansible` — ваш текущий рабочий каталог. Также можно удалить каталог `./group_vars`, потому что он нам больше не понадобится. Наконец, создайте два сценария — `create-users1.yml` и `create-users2.yml`, как показано в следующих листингах:

```
create-users1.yml
```

```
create-users2.yml
```

```

---
- name: Create users on webserver
  hosts: webserver
  become: yes

  tasks:
    - name: Create the 'webuser' account
      user:
        name: webuser
        state: present

    - name: Create the 'webadmin' account
      user:
        name: webadmin
        state: present

    - name: Create the 'webdev' account
      user:
        name: webdev
        state: present

```

```

---
- name: Create users on webserver
  hosts: webserver
  become: yes

  vars:
    users:
      - webuser
      - webadmin
      - webdev

  tasks:
    - name: Create the '{{ item }}'
      account
      user:
        name: "{{ item }}"
        state: present
      loop: "{{ users }}"

```

Оба сценария создают три учетные записи: `webuser`, `webadmin` и `webdev`. В сценарии `create-users1.yml` для этого есть три отдельные задачи, по одной на каждого пользователя. А `create-users2.yml` трижды повторяет одну задачу с помощью цикла `loop`:

```
loop: "{{ users }}"
```

Этот цикл перебирает элементы списка `users`, который определен как переменная сценария в предыдущих строках. Задача `user` на каждой итерации подставляет очередное имя пользователя в переменную `{{ item }}`.

- Прежде чем запускать любой из этих сценариев, давайте создадим еще один сценарий, который удаляет учетные записи. Назовите его `delete-users2.yml` и внесите в него код, похожий на `create-users2.yml`:

```

---
- name: Delete users on webserver
  hosts: webserver
  become: yes

  vars:
    users:
      - webuser
      - webadmin
      - webdev

  tasks:
    - name: Delete user
      user:
        name: "{{ item }}"
        state: absent
        remove: yes
        force: yes
      loop: "{{ users }}"

```

3. Теперь запустите сценарий `create-users1.yml`, нацелив его только на веб-сервер `ans-web1`. Вывод показывает, что было выполнено три задачи, по одной для каждого пользователя:

```
packt@neptune:~/ansible$ ansible-playbook create-users1.yml --limit ans-web1
PLAY [Create users on webservers] *****
TASK [Gathering Facts] *****
ok: [ans-web1]
TASK [Create the 'webuser' account] *****
ok: [ans-web1]
TASK [Create the 'webadmin' account] *****
changed: [ans-web1]
TASK [Create the 'webdev' account] *****
changed: [ans-web1]
PLAY RECAP *****
ans-web1 : ok=4 changed=2 unreachable=0 failed=0 skipped=0 rescued=0
ignored=0
```

4. Давайте удалим пользователей с помощью сценария `delete-users2.yml`. Обратите внимание, что на этот раз вывод отражает всего одну задачу, которая перебирает всех пользователей:

```
packt@neptune:~/ansible$ ansible-playbook delete-users2.yml --limit ans-web1
PLAY [Delete users on webservers] *****
TASK [Gathering Facts] *****
ok: [ans-web1]
TASK [Delete user] *****
changed: [ans-web1] => (item=webuser)
changed: [ans-web1] => (item=webadmin)
changed: [ans-web1] => (item=webdev)
PLAY RECAP *****
ans-web1 : ok=2 changed=1 unreachable=0 failed=0 skipped=0 rescued=0
ignored=0
```

5. Теперь запустите сценарий `create-users2.yml`, снова нацеленный только на `ans-web1`. Как и в предыдущем примере, все операции с учетными записями происходят в цикле в рамках одной задачи:

```
packt@neptune:~/ansible$ ansible-playbook create-users2.yml --limit ans-web1
PLAY [Create users on webservers] *****
TASK [Gathering Facts] *****
ok: [ans-web1]
TASK [Create the '{{ item }}' account] *****
ok: [ans-web1] => (item=webuser)
ok: [ans-web1] => (item=webadmin)
```

```
ok: [ans-web1] => (item=webdev)
```

```
PLAY RECAP *****
ans-web1 : ok=2 changed=0 unreachable=0 failed=0 skipped=0 rescued=0
ignored=0
```

Между двумя сценариями для создания пользователей есть существенная разница:

- Первый сценарий выполняет задачи для каждого пользователя в отдельности. Хотя дублирование задачи — не затратная операция, тем не менее среда выполнения Ansible испытает значительную нагрузку, если придется создавать сотни пользователей.
- Во втором сценарии запускается всего одна задача, которая трижды вызывает модуль `user` для каждой учетной записи. Загрузить модуль — гораздо менее затратная операция, чем выполнить задачу.

Чтобы получить дополнительную информацию о циклах, обратитесь к официальной документации: docs.ansible.com/ansible/latest/user_guide/playbooks_loops.html.

Теперь, когда вы знаете, как реализовать простой цикл, давайте сделаем так, чтобы наш сценарий был компактнее и чтобы его было удобнее сопровождать.

Как оптимизировать сценарии Ansible

В этом разделе мы не только усовершенствуем наши сценарии, но и попытаемся приблизиться к практике из реального мира, когда учетные записи и их пароли хранятся в защищенном формате и их удобно использовать повторно.

Давайте разместим сведения о пользователях в файле `users.yml`, а пароли — в файле `users_passwords.yml`:

```
users.yml
---
# Пароли хранятся в файле
# users_passwords.yml
```

```
webusers:
- username: webuser
  comment: Обычный пользователь
- username: webadmin
  comment: Администратор веб-сервера
- username: webdev
  comment: Веб-разработчик
```

```
users_passwords.yml
---
# Имена пользователей
# соответствуют файлу users.yml
# Порядок не имеет значения
```

```
webuser:
  password: bb37e5d1
webadmin:
  password: 7785b8a4
webdev:
  password: 8365b176
```

Рассмотрим эти файлы подробнее:

- Файл `users.yml` содержит одну пару «ключ — значение»:

Ключ	<code>webusers</code>
Значение	Список пар с ключами <code>username</code> и <code>comment</code>

- Файл `users_passwords.yml` содержит три пары:

Ключи	Имена пользователей (<code>webuser</code> , <code>webadmin</code> , <code>webdev</code>)
Значения	Вложенные словари с парами <code>password: значение</code>

Мы можем обновить файл `users_passwords.yml` с помощью команды `ansible-vault edit` или создать новый файл с нуля. Во втором случае нужно зашифровать файл, как описывалось ранее в разделе «Как работать с секретами».

Новый сценарий `create-users.yml` выглядит так:

```
---
- name: Create users on webserver
  hosts: webserver
  become: yes

  tasks:
    - name: Load users
      include_vars:
        file: users.yml
        name: users

    - name: Load passwords
      include_vars:
        file: users_passwords.yml
        name: passwords

    - name: Create user accounts
      user:
        name: "{{ item.username }}"
        comment: "{{ item.comment }}"
        password:
          "{{ passwords[item.username]['password'] | password_hash('sha512') }}"
      loop: "{{ users.webusers }}"
      update_password: always
      state: present
```

Давайте разберем этот сценарий. В нем три задачи:

- **Load users** (Загрузить учетные записи). Считывает информацию о пользователях из файла `users.yml` и сохраняет соответствующие значения в словаре `users`.
- **Load passwords** (Загрузить пароли). Считывает пароли из зашифрованного файла `passwords.yml` и сохраняет их в словаре `passwords`.
- **Create user accounts** (Создать учетные записи). Перебирает элементы списка `users.webusers` и для каждого элемента создает учетную запись с соответствующими параметрами. Задача ищет пароли в словаре `passwords`, опираясь на `item.username`.

Перед тем как запускать сценарий, зашифруйте файл `users_passwords.yml` с помощью такой команды:

```
ansible-vault encrypt users_passwords.yml
```

Теперь запустите сценарий:

```
packt@neptune:~/ansible$ ansible-playbook --ask-vault-pass create-users.yml
Vault password:

PLAY [Create users on webservers] *****

TASK [Gathering Facts] *****
ok: [ans-web1]
ok: [ans-web2]

TASK [Load users] *****
ok: [ans-web1]
ok: [ans-web2]

TASK [Load passwords] *****
ok: [ans-web1]
ok: [ans-web2]

TASK [Create user accounts] *****
changed: [ans-web1] =>
    (item={'username': 'webuser', 'comment': 'Обычный пользователь'})
changed: [ans-web2] =>
    (item={'username': 'webuser', 'comment': 'Обычный пользователь'})
changed: [ans-web1] =>
    (item={'username': 'webadmin', 'comment': 'Администратор веб-сервера'})
changed: [ans-web2] =>
    (item={'username': 'webadmin', 'comment': 'Администратор веб-сервера'})
changed: [ans-web1] =>
    (item={'username': 'webdev', 'comment': 'Веб-разработчик'})
changed: [ans-web2] =>
    (item={'username': 'webdev', 'comment': 'Веб-разработчик'})

PLAY RECAP *****
ans-web1 : ok=4 changed=1 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0
ans-web2 : ok=4 changed=1 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0
```

Помимо трех описанных задач, в выводе фигурирует задача **Gathering Facts** (Собрать факты), которая обнаруживает управляемые узлы и связанные с ними системные переменные (так называемые **факты**). Мы познакомимся с фактами Ansible позже в этой главе.

Чтобы проверить новые учетные записи, можно использовать методы, которые описывались ранее в разделе «Как работать с секретами». В качестве упражнения создайте сценарий `delete-users.yml`, который удаляет учетные записи по той же схеме, по которой `create-users.yml` их создает.

Теперь давайте посмотрим, как еще можно улучшить наш сценарий, чтобы с его помощью можно было прозрачно создавать пользователей на всех узлах из

описи, будь то веб-серверы или базы данных. Для этого мы задействуем задачи с условиями.

Задачи с условиями

Условные конструкции в сценариях Ansible запускают задачи в зависимости от тех или иных условий (или состояний). Условием может быть, например, значение **переменной** или **факта** либо **результат выполнения** предыдущей задачи. Условия в Ansible объявляются с помощью директивы **when** на уровне задачи.

Вы уже знаете, что такое переменные и как использовать их в сценариях. Факты и результаты — это, по сути, особые виды переменных, которые мы рассмотрим в контексте задач с условиями.

Факты в Ansible

Факты — это переменные, которые предоставляют определенные сведения об удаленных управляемых узлах. Имена фактов начинаются с префикса **ansible_**, например:

<code>ansible_distribution</code>	Дистрибутив ОС (например, Ubuntu)
<code>ansible_all_ipv4_addresses</code>	Адреса IPv4
<code>ansible_architecture</code>	Архитектура платформы (например, x86_64 или i386)
<code>ansible_processor_cores</code>	Количество ядер процессора
<code>ansible_memfree_mb</code>	Доступная память (в мегабайтах)

Если помните, раньше мы разбивали множество узлов на группы **ubuntu** и **debian** в зависимости от того, какая система на них установлена. Но чтобы выполнять разные задачи для разных дистрибутивов (а не только для двух упомянутых), можно не группировать узлы в явном виде, а собирать факты об управляемых узлах и автоматически определять тип их ОС.

Например, давайте создадим сценарий, который обновляет систему с помощью механизмов, специфичных для той или иной платформы. Предположим, что на одних управляемых узлах установлена Ubuntu, а на других — Fedora. Назовите сценарий **install-updates.yml** и добавьте в него такое содержимое:

```
---
- name: Install system updates
  hosts: all
  become: yes

tasks:

- name: Install Ubuntu system updates
  apt: upgrade=dist update_cache=yes
  when: ansible_distribution == "Ubuntu"
```

```
- name: Install Fedora system updates
  dnf: name=* state=latest update_cache=yes
  when: ansible_distribution == "Fedora"
```

Сценарий нацелен на все узлы и содержит две задачи, каждая из которых запускается в зависимости от значения факта `ansible_distribution`: первая — только на узлах Ubuntu, а вторая — только на узлах Fedora.

Запустите сценарий обычным способом:

```
ansible-playbook install-updates.yml
```

Если для узлов есть обновления, которые нужно установить, сценарий может выполняться довольно долго.

Далее мы рассмотрим, как использовать в задачах с условиями переменные Ansible, специфичные для окружения.

Магические переменные

Магические переменные описывают локальное окружение Ansible и его конфигурацию, например:

ansible_playhosts	Список активных узлов в текущей группе задач
group_names	Список всех групп, в которые входит текущий узел
vars	Словарь со всеми переменными в текущей группе задач
ansible_version	Версия Ansible

Чтобы увидеть, чем полезны магические переменные в задачах с условиями, мы в очередной раз усовершенствуем сценарий `create-users.yml` так, чтобы он создавал определенные наборы пользователей на разных группах узлов. До сих пор сценарий создавал учетные записи `webuser`, `webadmin` и `webdev` только на веб-серверах — то есть узлах, которые входят в группу `webservers` (`ans-web1` и `ans-web2`). А что, если нужно создать аналогичный набор пользователей — `dbuser`, `dbadmin` и `dbdev` — на всех серверах баз данных? Это можно сделать так:

1. Добавьте сведения о новых учетных записях и их пароли в файлы `users.yml` и `users_passwords.yml` соответственно. Вот как должны теперь выглядеть эти файлы:

```
users.yml
---
# Пароли хранятся в файле
# users_passwords.yml

webusers:
  - username: webuser
    comment: Обычный пользователь
```

```
users_passwords.yml
---
# Имена пользователей
# соответствуют файлу users.yml
# Порядок не имеет значения

# Пароли пользователей веб-сервера

webuser:
  password: bb37e5d1
```

```

- username: webadmin
  comment: Администратор веб-сервера
webadmin:
  password: 7785b8a4

- username: webdev
  comment: Веб-разработчик
webdev:
  password: 8365b176

dbusers:
  # Пароли пользователей БД

- username: dbuser
  comment: Обычный пользователь БД
dbuser:
  password: 4695b3db

- username: dbadmin
  comment: Администратор БД
dbadmin:
  password: 99857ee9

- username: dbdev
  comment: Разработчик БД
dbdev:
  password: a966ada8

```

Обратите внимание, что файл `users_passwords.yml` можно редактировать с помощью команды `ansible-vault edit`, а можно дешифровать, отредактировать и заново зашифровать.

2. Теперь давайте напишем новый сценарий с условиями для того, чтобы обрабатывать группы `webusers` и `databases` по отдельности. Создайте файл `create-users3.yml` с таким содержимым:

```

---
- name: Create users
  hosts: all
  become: yes
  tasks:

  - name: Load users
    include_vars:
      file: users.yml
      name: users

  - name: Load passwords
    include_vars:
      file: users_passwords.yml
      name: passwords

  - name: Create web user accounts
    user:
      name: "{{ item.username }}"
      comment: "{{ item.comment }}"
      password:
        "{{ passwords[item.username]['password'] | password_hash('sha512') }}"
      update_password: always
      state: present
    loop: "{{ users.webusers }}"
    when: "'webservers' in group_names"

  - name: Create database user accounts
    user:

```

```

name: "{{ item.username }}"
comment: "{{ item.comment }}"
password:
  "{{ passwords[item.username][ 'password' ] | password_hash('sha512') }}"
update_password: always
state: present
loop: "{{ users.dbusers }}"
when: "'databases' in group_names"

```

3. Запустите сценарий:

```
ansible-playbook --ask-vault-pass create-users3.yml
```

Вот фрагмент вывода, который показывает, что задача, связанная с веб-серверами, не обрабатывает серверы баз данных. Можете самостоятельно убедиться, что задача, связанная с серверами баз данных, аналогично пропускает веб-серверы.

```

TASK [Create web user accounts] *****

skipping: [ans-db1] =>
  (item={'username': 'webuser', 'comment': 'Обычный пользователь'})
# ... аналогично для пользователей webadmin и webdev ...

skipping: [ans-db1]
# ... аналогично для узла db2 ...

changed: [ans-web1] => changed: [ans-web1] =>
  (item={'username': 'webuser', 'comment': 'Обычный пользователь'})
# ... далее аналогично листингу на с. 652 ...

```

Полный список специальных переменных Ansible, включая магические переменные, можно найти на странице docs.ansible.com/ansible/latest/reference_appendices/special_variables.html, а дополнительные сведения о фактах и магических переменных — на странице docs.ansible.com/ansible/latest/playbook_guide/playbooks_vars_facts.html.

Далее мы рассмотрим регистровые переменные, с помощью которых можно отслеживать результаты выполнения задач.

Регистровые переменные

Регистры Ansible фиксируют вывод задачи в переменных, которые называются **регистровыми**. Ansible захватывает вывод в переменную с помощью директивы `register`. Типичный пример того, как используются регистровые переменные, можно пронаблюдать, если требуется собирать результаты выполнения задачи для целей отладки. В более сложных рабочих процессах задачи могут запускаться или не запускаться в зависимости от результатов предыдущих задач.

Давайте рассмотрим гипотетический пример. По мере того как мы набираем новых пользователей и создаем различные учетные записи на всех серверах, мы хотим убедиться, что количество пользователей не превышает максимально

допустимого. Если превышает, придется принять меры: запустить новый сервер, перераспределить пользователей и т. д. Чтобы проверить их количество, можно сделать так:

1. Сначала создайте сценарий `count-users.yml` с таким содержанием:

```
- name: Узнать, превышает ли количество пользователей максимально допустимое
  hosts: all
  become: yes

  vars:
    max_allowed: 30

  tasks:

    - name: Подсчитать количество пользователей
      shell: "getent passwd | wc -l"
      register: count

    - name: Вывести текущее и максимально допустимое количество пользователей
      debug:
        msg: "Количество пользователей: {{ count.stdout }} из {{ max_allowed }}"

    - name: Сообщить, что превышено максимально допустимое кол-во пользователей
      debug:
        msg: "Превышено максимально допустимое количество пользователей!"
        when: count.stdout | int > max_allowed
```

В этом сценарии три задачи:

- **Подсчитать количество пользователей.** Эта задача подсчитывает количество пользователей с помощью модуля `shell` и регистрирует переменную `count`, в которую заносится вывод задачи.
- **Вывести текущее и максимально допустимое количество пользователей.** Простая задача для отладки, которая выводит данные в терминал.
- **Сообщить, что превышено максимально допустимое количество пользователей.** Задача с условием, которая запускается, если пользователей слишком много. Задача проверяет значение регистровой переменной `count` и сравнивает его с `max_allowed`.
- Последнюю строку в нашем сценарии стоит пояснить. В ней фигурирует `count.stdout` — значение регистровой переменной для стандартного вывода. Это строковые данные, и мы преобразовываем их в целое число с помощью `count.stdout | int`, а затем сравниваем это число с `max_allowed`.

2. Запустите сценарий на узле `ans-web1`:

```
ansible-playbook count-users.yml --limit ans-web1
```

В выводе будут такие строки:

```
TASK [Вывести текущее и максимально допустимое количество
пользователей] *****
```

```

ok: [ans-web1] => {
  "msg": "Количество пользователей: 36 из 30"
}

TASK [Сообщить, что превышено максимально допустимое количество
пользователей] ****
ok: [ans-web1] => {
  "msg": "Превышено максимально допустимое количество пользователей!"
}

```

В нашем случае оказалось 36 пользователей, а это больше максимального значения 30. Как видите, задача сработала так, как и должна была.

3. Теперь давайте изменим максимальное количество пользователей в сценарии `count-users.yml`:

```
max_allowed: 50
```

4. Сохраните и повторно запустите сценарий. На этот раз вывод покажет, что соответствующая задача пропущена:

```

TASK [Сообщить, что превышено максимально допустимое количество
пользователей] ****
skipping: [ans-web1]

```

Чтобы больше узнать о задачах с условиями в сценариях Ansible, посетите страницу docs.ansible.com/ansible/latest/playbook_guide/playbooks_conditionals.html. Если комбинировать условия с фактами и специальными переменными Ansible, можно писать чрезвычайно мощные сценарии и автоматизировать многие операции системного администрирования.

В следующих разделах мы поговорим о том, как еще можно сделать наши сценарии универсальнее и оптимизировать их для повторного использования. Прежде всего мы рассмотрим шаблоны динамической конфигурации.

Как использовать шаблоны Jinja2 в сценариях Ansible

Одна из самых распространенных задач управления конфигурацией — копировать файлы на управляемые узлы. Для этого в Ansible служит модуль `copy`. Типичная операция копирования файлов в сценариях Ansible выглядит так:

```

- copy:
  src: motd
  dest: /etc/motd

```

Эта задача копирует исходный файл `motd` в каталог `/etc/motd` на удаленном узле. Хотя эта схема позволяет легко скопировать статические файлы на несколько узлов, она не сможет модифицировать их «на лету».

Возьмем, например, файл сетевой конфигурации, который содержит IP-адрес узла. Если скопировать этот файл на все узлы, чтобы настроить на них сетевые параметры, то все узлы, кроме одного, могут оказаться недоступными. В идеале

в конфигурационном файле должны быть *заполнители* для динамического содержимого (например, IP-адресов), которые заменяются на конкретные значения в зависимости от целевого узла.

Эту функциональность в Ansible обеспечивает модуль `template` с шаблонизатором **Jinja2**, который использует конструкции языка Python для переменных и выражений в шаблоне. Синтаксис модуля `template` очень похож на `copy`:

```
- template:
  src: motd.j2
  dest: /etc/motd
```

В данном случае источник (`motd.j2`) — это файл шаблона Jinja2 с настройками для конкретного узла. Перед тем как скопировать файл на удаленный узел, Ansible считывает шаблон и заменяет динамическое содержимое на данные, специфичные для узла. Эта обработка происходит на управляющем узле Ansible.

Чтобы проиллюстрировать некоторые преимущества шаблонов Ansible и разобраться, как они работают, мы рассмотрим пару случаев из практики и создадим для каждого из них шаблон Jinja2, а затем напомним и запустим соответствующие сценарии Ansible.

В этом разделе мы создадим шаблоны для двух файлов:

- **Сообщение дня** (`motd`¹), которое отображает особое уведомление для пользователей о плановом обслуживании системы.
- **Файл узлов** (`hosts`), который на каждом управляемом узле находится в расположении `/etc/hosts` и содержит имена других управляемых узлов.

Начнем с шаблона для сообщения дня.

Шаблон сообщения дня

В начале этого раздела мы использовали файл `/etc/motd` в качестве примера. В Linux содержимое этого файла отображается, когда пользователь входит в терминал. Допустим, вы планируете обновлять свои веб-серверы ночью с четверга на пятницу и хотели бы по-дружески напомнить пользователям о предстоящем отключении. Ваше сообщение может быть примерно таким:

Этот сервер будет отключен для обслуживания ночью с четверга на пятницу.

В таком сообщении нет ничего особенного, и соответствующий файл `motd` можно легко развернуть с помощью простой задачи `copy`. Возможно, в большинстве случаев этого будет достаточно, если только вас не смущает, что пользователи могут запутаться в том, что такое «этот сервер». Также стоит учесть, что ночь с четверга на пятницу в вашем часовом поясе может оказаться днем пятницы на другом конце света, поэтому лучше, если сообщение будет конкретнее.

¹ Сокр. Message of the day (сообщение дня). — *Примеч. пер.*

Скорее всего, для веб-сервера `ans-web1` лучше подойдет такая формулировка:

Сервер `ans-web1` (172.16.191.12) будет отключен для обслуживания 8 апреля 2021 (чт), между 02:00 и 03:00 (UTC-08:00).

Соответственно, на веб-сервере `ans-web2` сообщение будет отражать другой IP-адрес и имя узла. В идеале шаблон должен подходить для многократного использования в разных часовых поясах, а сценарии должны запускаться на глобально распределенных управляющих узлах Ansible. Давайте посмотрим, как реализовать такой шаблон (будем считать, что ваш текущий рабочий каталог — `~/ansible`):

1. Сначала создайте каталог `templates` в локальном каталоге проекта Ansible:

```
mkdir -p ~/ansible/templates
```

Ansible будет искать файлы шаблонов либо в локальном каталоге, где находится сценарий, либо в каталоге `./templates`.

2. В текстовом редакторе создайте в каталоге `./templates` файл `motd.j2` с таким содержанием:

```
1  {# Эти переменные передаются из сценария #}
2  {% set date = '2021-04-08' %}
3  {% set start_time = date ~ 'T02:00:00-0800' %}
4  {% set end_time = date ~ 'T03:00:00-0800' %}

6  {% set fmt = '%Y-%m-%dT%H:%M:%S%z' %}
7  {% set start = (start_time | to_datetime(format=fmt)) %}
8  {% set end = (end_time | to_datetime(format=fmt)) %}

10 {# Мы также используем специальную переменную {{ inventory_hostname }} #}
11 Сервер {{ ansible_facts.fqdn }} ({{ ansible_facts.default_ipv4.address }})
12 будет отключен для обслуживания {{ start.strftime('%A, %B %-d, %Y' ) }},
13 между {{ start.strftime('%-I' ) }} и {{ end.strftime('%-I %p (%Z)' ) }}.
```

Обратите внимание на некоторые компоненты синтаксиса Jinja2:

Комментарии	{# ... #}
Выражения	{% ... %}
Ссылки на внешние переменные	{{ ... }}

Вот что делает этот сценарий:

- *Строки 1–4* определяют начальный набор локальных переменных, в которых хранятся временные границы отключения: дата (`date`), время начала (`start_time`) и время окончания (`end_time`).
- *В строке 6* задается нужный формат даты и времени (`fmt`) для этих переменных.
- *В строках 7–8* создаются объекты Python типа `datetime`, которые соответствуют начальному и конечному времени и форматируются так, как они будут выводиться в сообщении для пользователей.

- В строках 11–13 выводится сообщение с интервалом времени и двумя фактами Ansible: **полным доменным именем** (`ansible_facts.fqdn`) и адресом IPv4 (`ansible_facts.default_ipv4.address`) того узла, на котором отображается сообщение.
3. Теперь давайте создадим сценарий, который запускает этот шаблон. Назовите сценарий `update-motd.yml` и добавьте в него такое содержимое:

```
---
- name: Обновить сообщение дня
  hosts: all
  become: yes
  tasks:
    - name: Развернуть шаблон motd
      template:
        src: motd.j2
        dest: /etc/motd
        owner: root
        group: root
        mode: 0644
```

Модуль `template` обрабатывает файл `motd.j2` и генерирует необходимое динамическое содержимое, а затем копирует файл в каталог `/etc` на удаленном узле с необходимыми правами доступа.

4. Теперь мы готовы запустить сценарий:

```
packt@neptune:~/ansible$ ansible-playbook update-motd.yml
```

```
PLAY [Обновить сообщение дня] *****
```

```
TASK [Gathering Facts] *****
```

```
ok: [ans-web1]
ok: [ans-db1]
ok: [ans-web2]
ok: [ans-db2]
```

```
TASK [Развернуть шаблон motd] *****
```

```
changed: [ans-web1]
changed: [ans-db1]
changed: [ans-web2]
changed: [ans-db2]
```

```
PLAY RECAP *****
```

```
ans-db1  : ok=2 changed=1 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0
ans-db2  : ok=2 changed=1 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0
ans-web1 : ok=2 changed=1 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0
ans-web2 : ok=2 changed=1 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0
```

5. С помощью следующей команды можно немедленно проверить, как сообщение `motd` отображается на любом из узлов (например, `ans-web1`):

```
ansible ans-web1 -a "cat /etc/motd"
```

Предыдущая команда запускается удаленно на узле `ans-web1` и отображает содержимое файла `/etc/motd`:

```
packt@neptune:~/ansible$ ansible ans-web1 -a "cat /etc/motd"
ans-web1 | CHANGED | rc=0 >>
```

Сервер `ans-web1` (172.16.191.12) будет отключен для обслуживания 8 апреля 2021 (чт), между 02:00 и 03:00 (UTC-08:00).

6. Также можно подключиться по SSH к любому из узлов и убедиться, что там отображается нужное сообщение дня:

```
packt@neptune:~/ansible$ ssh packt@ans-web1
```

Сервер `ans-web1` (172.16.191.12) будет отключен для обслуживания 8 апреля 2021 (чт), между 02:00 и 03:00 (UTC-08:00).

Last login: Tue Apr 8 00:46:14 2021 from 192.168.122.1

```
packt@ans-web1 :~$
```

7. Теперь, когда мы умеем работать с шаблонами Ansible, давайте доработаем файл `motd.j2`, чтобы он лучше подходил для повторного использования. Мы *параметризуем* шаблон: заменим жестко закодированные локальные переменные для даты и времени на входные переменные, которые передаются из сценария. В результате шаблон можно будет использовать с разными сценариями и разными периодами обслуживания. Замените строки 1–8 на такой код:

```
{% set start_time_ = date ~ 'T' ~ start_time ~ utc %}
{% set end_time_ = date ~ 'T' ~ end_time ~ utc %}

{% set fmt = '%Y-%m-%dT%H:%M:%S%z' %}
{% set start = (start_time_ | to_datetime(format=fmt)) %}
{% set end = (end_time_ | to_datetime(format=fmt)) %}
```

Значимые изменения находятся в первых двух строках, где мы формируем *локальные переменные* `start_time_` и `end_time_` (с суффиксом `_`) из соответствующих *входных переменных* `start_time` и `end_time` (без суффикса). Вы можете использовать и другие схемы именования, которые совместимы с Ansible.

8. Теперь отредактируйте блок `vars` в сценарии `update-motd.yml`:

```
...
become: yes
vars:
  date: "2021-04-08"
  start_time: "02:00:00"
  end_time: "03:00:00"
  utc: "-0800"
tasks:
  ...
```

Если запустить обновленный сценарий, он должен привести к тому же результату, что и предыдущая реализация. Попробуйте выполнить это упражнение самостоятельно.

Далее мы рассмотрим еще один пример развертывания на основе шаблонов: напишем сценарий, который обновляет файл `/etc/hosts` на управляемых узлах и вносит туда сведения обо всех остальных серверах в группе.

Шаблон файла `hosts`

С помощью шаблонов Jinja2 также можно автоматически обновлять файлы на управляемых узлах, например файл `/etc/hosts`. Он содержит IP-адреса и имена всех узлов в сети, и типичная задача системного администратора состоит в том, чтобы поддерживать его в актуальном состоянии. В этом упражнении мы создадим новый шаблон для файла `hosts`, а также файл YAML, в котором будет использоваться этот шаблон. Выполните такие действия:

1. Создайте новый файл шаблона `hosts.j2` в каталоге `~/ansible/templates` и добавьте в него такое содержимое:

```
127.0.0.1 {{ inventory_hostname }} localhost

{% for host in groups ['all'] %}
{% if host != inventory_hostname %}
{{ hostvars[host].ansible_facts.default_ipv4.address }} {{ host }}
{% endif %}
{% endfor %}
```

Вот что делает этот шаблон:

- Добавляет запись `localhost` для текущего узла, на который ссылается специальная переменная Ansible `inventory_hostname`.
 - В цикле перебирает все узлы в описи; на их список ссылается еще одна специальная переменная `groups['all']`.
 - На каждой итерации цикла проверяет, совпадает ли текущий узел с целевым, и выполняет следующее действие, только если узлы *различаются*.
 - Добавляет новую запись об узле, которая состоит из его адреса IPv4 по умолчанию (`default_ipv4.address`) и имени узла (`host`).
2. Теперь давайте создадим файл сценария `update-hosts.yml`, который использует шаблон `hosts.j2`. Добавьте в сценарий такое содержание:

```
---
- name: Обновить файл hosts
  hosts: all
  become: yes
  tasks:
    - name: Развернуть шаблон файла hosts
      template:
        src: hosts.j2
        dest: /etc/hosts
        owner: root
        group: root
        mode: 0644
```

Этот сценарий очень похож на `update-motd.yml` и нацелен на файл `/etc/hosts`.

- После того как сценарий и файл шаблона готовы, запустите сценарий:

```
ansible-playbook update-hosts.yml
```

- После этого можно проверить файл `/etc/hosts` на любом узле (например, `ans-web1`) с помощью такой команды:

```
packt@neptune:~/ansible$ ansible ans-web1 -a "cat /etc/hosts"
ans-web1 | CHANGED | rc=0 >>
127.0.0.1 ans-web1 localhost

192.168.122.147 ans-web2
192.168.122.254 ans-db1
192.168.122.25 ans-db2
```

Как видите, здесь отображаются ожидаемые записи обо всех узлах.

- Кроме того, можно подключиться по SSH к любому узлу (например, `ans-web1`) и с помощью `ping` опросить любой другой узел по имени (например, `ans-db2`):

```
packt@neptune:~/ansible$ ssh packt@ans-web1

packt@ans-web1:~ $ ping ans-db2
PING ans-db2 (192.168.122.25) 56(84) bytes of data.
64 bytes from ans-db2 (192.168.122.25): icmp_seq=1 ttl=64 time=1.41 ms
64 bytes from ans-db2 (192.168.122.25): icmp_seq=1 ttl=64 time=0.914 ms
...
```

На этом мы заканчиваем тему шаблонов Ansible. Однако все, что мы успели рассмотреть в этом разделе, — лишь малая часть мощных возможностей, которые присущи шаблонам Jinja2. Мы рекомендуем вам ознакомиться с официальной документацией (docs.ansible.com/ansible/latest/playbook_guide/playbooks_templating.html), а также с источниками, которые перечислены в разделе «Дополнительная литература» в конце этой главы.

Давайте теперь обратим внимание на еще одну важную особенность современных платформ управления конфигурациями: совместный доступ к повторно используемым и гибким модулям, которые решают различные задачи системного администрирования. Для этого Ansible предоставляет доступную и расширяемую структуру — **роли Ansible** и **Ansible Galaxy**. В следующем разделе вы узнаете, что такое роли и как они помогают повторно использовать автоматизацию.

Роли Ansible

Роли Ansible позволяют объединить автоматизированные рабочие процессы в единицы многократного использования. Фактически роль — это пакет со сценариями и другими ресурсами, которые адаптированы к определенной конфигурации с помощью переменных. Чтобы вызвать роль, сценарий может пере-

дать необходимые параметры и запустить ее, как любую другую задачу. С функциональной точки зрения роли инкапсулируют обобщенную процедуру управления конфигурацией, и благодаря этому их можно повторно применять в различных проектах и даже использовать совместно с другими коллегами.

Вот основные преимущества ролей:

- Инкапсуляция обеспечивает автономный пакет, который можно легко передать кому-нибудь. Инкапсуляция также позволяет **разделить обязанности**: несколько специалистов по DevOps и системных администраторов могут разрабатывать роли параллельно.
- Роли помогают сделать крупные проекты автоматизации более управляемыми.

В этом разделе мы расскажем о том, как создать роль и как использовать ее на примере сценария.

Создавая роли, мы обычно следуем таким принципам:

- Создайте или инициализируйте структуру каталогов для роли. Эти каталоги должны содержать все ресурсы, которые необходимы для роли, в хорошо организованном виде.
- Реализуйте содержание роли. Создайте соответствующие сценарии, файлы, шаблоны и т. д.
- Всегда начинайте с простой функциональности и постепенно переходите к более продвинутой. Тестируйте свои сценарии по мере того, как добавляете новый код.
- Добивайтесь, чтобы ваша реализация была как можно универсальнее. Храните настройки в переменных.
- Не храните конфиденциальные данные в сценариях или связанных с ними файлах. Обеспечьте входные параметры для таких данных.
- Создайте фиктивный сценарий с простой группой команд, который запускает вашу роль, и тестируйте ее с его помощью.
- Создавайте свою роль с расчетом на то, что ее будут использовать другие люди. Сделайте ее простой в использовании и поделитесь ею с коллегами, если считаете, что она принесет пользу сообществу.

В общих чертах создание роли состоит из таких этапов:

1. Инициализировать структуру каталогов роли.
2. Создать содержание роли.
3. Протестировать роль.

Для того чтобы попрактиковаться в создании роли, мы возьмем за основу сценарий `create-users3.yml`, который создали ранее в разделе «Магические переменные». Скопируйте этот файл под новым именем, например, `create-users-`

`role.yml`. Прежде чем двигаться дальше, давайте добавим такую строку в раздел `[defaults]` файла `ansible.cfg`, который находится в каталоге `~/ansible`:

```
roles_path = ~/ansible
```

Этот параметр конфигурации задает местоположение по умолчанию для наших ролей.

Теперь давайте инициализируем каталог роли.

Как инициализировать структуру каталогов роли

Ansible предъявляет строгие требования к структуре каталогов роли. Корневой каталог должен называться так же, как и роль, например `create-users-role`. Этот каталог можно создать вручную или с помощью специальной утилиты командной строки `ansible-galaxy`, которая позволяет управлять ролями.

Чтобы создать основу нашего каталога роли, запустите такую команду:

```
packt@neptune:~/ansible$ ansible-galaxy init create-users-role
- Role create-users-role was created successfully
```

Структуру каталогов можно вывести с помощью команды `tree`. Если она не установлена в вашей системе, установите ее с помощью стандартной системы управления пакетами:

```
packt@neptune:~/ansible$ tree
```

```
...
├── create-users-role
│   ├── defaults
│   │   └── main.yml
│   ├── files
│   ├── handlers
│   │   └── main.yml
│   ├── meta
│   │   └── main.yml
│   ├── README.md
│   ├── tasks
│   │   └── main.yml
│   ├── templates
│   ├── tests
│   │   ├── inventory
│   │   └── test.yml
│   └── vars
│       └── main.yml
└── create-users-role.yml
...
```

Давайте разберемся, для чего служит каждый каталог и соответствующий файл YAML в каталоге роли:

- `defaults/main.yml` — переменные по умолчанию для роли. У них самый низкий приоритет среди всех доступных переменных, и их можно переопределить любыми другими переменными.

- `files` — статические файлы, которые используются в задачах роли.
- `handlers/main.yml` — обработчики, которые использует роль. Обработчики — это задачи, которые запускаются другими задачами; подробнее о них можно прочитать на странице docs.ansible.com/ansible/latest/playbook_guide/playbooks_handlers.html.
- `README.md` — справочный документ о том, зачем нужна роль и как ее использовать.
- `meta/main.yml` — метаданные о роли: автор, модель лицензирования, платформы, зависимости от других ролей и пр.
- `tasks/main.yml` — задачи, которые выполняет роль.
- `templates` — файлы шаблонов, на которые ссылается роль.
- `tests/test.yml` — шаблон для тестирования роли. В каталоге `tests` также может размещаться образец файла описи.
- `vars/main.yml` — переменные, которые используются внутри роли. Эти переменные имеют высокий приоритет, и их нельзя изменить или переопределить.

Теперь, когда мы знакомы с каталогом роли и соответствующими файлами ресурсов, давайте создадим нашу первую роль.

Как создать содержание роли

Следуя общей практике, мы возьмем ранее созданный сценарий `create-users.yml` и превратим его в роль `create-users-role.yml`.

Создайте в каталоге `./create-users-role` два новых файла, похожих на `users.yml` и `users_passwords.yml`, с которыми мы работали ранее:

```
users-role.yml
---
# Учетные записи для примера
# Пароли хранятся в файле
# users_passwords.yml

list:
  - username: testuser
    comment: Тестовый пользователь

  - username: testadmin
    comment: Тестовый администратор

  - username: testdev
    comment: Тестовый разработчик
```

```
user-passwords-role.yml
---
# Пароли для примера
# Имена пользователей
# соответствуют файлу users-role.yml
# Порядок не имеет значения

testuser:
  password: bb37e5d1

testadmin:
  password: 7785b8a4

testdev:
  password: 8365b176
```

Как вы могли заметить, мы присвоили учетным записям более общие имена, а также изменили ключ словаря с `webusers` на `list` в файле `users-role.yml`. Помните, что Ansible требует, чтобы в файлах YAML, которые предоставляют переменные, были ключи корневого уровня.

Вот как выглядит обновленный сценарий `create-users-role.yml`:

```
---
- name: Create users
  hosts: all
  become: yes

  vars:
    users_file: users-role.yml
    passwords_file: users-passwords-role.yml

  tasks:

    - name: Load users
      include_vars:
        file: "{{ users_file }}"
        name: users

    - name: Load passwords
      include_vars:
        file: "{{ passwords_file }}"
        name: passwords

    - name: Create user accounts
      user:
        name: "{{ item.username }}"
        comment: "{{ item.comment }}"
        password:
          "{{ passwords[item.username]['password'] | password_hash('sha512') }}"
      loop: "{{ users.list }}"
      update_password: always
      state: present
```

Мы внесли следующие изменения:

- Директива `loop` теперь считывает `users.list` вместо `users.webusers` согласно тому, как изменился соответствующий ключ словаря в файле `users.yml`.
- Мы вставили ссылки на файлы в директивах `include_vars`, чтобы использовать переменные вместо жестко заданных имен файлов.
- Мы добавили блок `vars`, в котором переменные `users_file` и `passwords_file` указывают на соответствующие файлы YAML.

После этих правок мы готовы реализовывать роль. Выполните такие операции в каталоге `create-users-role`:

1. Скопируйте переменные из блока `vars` файла `create-users-role.yml` в файл `defaults/main.yml`. В итоге этот файл должен выглядеть так:

```
---
# Файлы по умолчанию для роли create-users-role
users_file: users-role.yml
passwords_file: users-passwords-role.yml
```

2. Скопируйте задачи из `create-users-role.yml` в файл `tasks/main.yml`. Убедитесь, что вы сохранили отступы одних строк относительно других. Файл `main.yml` должен выглядеть так:

```
---
# Файл задач для роли create-users-role
- name: Load users
  include_vars:
    file: "{{ users_file }}"
    name: users

- name: Load passwords
  include_vars:
    file: "{{ passwords_file }}"
    name: passwords

- name: Create user accounts
  user:
    name: "{{ item.username }}"
    comment: "{{ item.comment }}"
    password:
      "{{ passwords[item.username]['password'] | password_hash('sha512') }}"
    update_password: always
    state: present
  loop: "{{ users.list }}"
```

3. В файле `tests/test.yml` создайте простой сценарий, который будет использовать роль:

```
---
- hosts: all
  become: yes
  roles:
    - role: create-users-role
      vars:
        users_file: users-role.yml
        passwords_file: users-passwords-role.yml
```

Скопируйте или переместите файлы `users-role.yml` и `users-passwords-role.yml` в каталог `tests`.

Мы также рекомендуем обновить файл `README.md` в каталоге `create-users-role` и написать в нем, зачем нужна роль и как ее использовать. Также упомяните о том, что для роли требуются файлы `users-role.yml` и `users-passwords-role.yml` с соответствующими структурами данных. Имена этих файлов можно изменить с помощью переменных `users_file` и `passwords_file` в файле `defaults/main.yml`. Также в `README.md` можно привести несколько примеров того, как использовать роль.

Давайте создадим еще один сценарий `test2.yml` с задачей, которая запускает роль:

```
---
- hosts: all
  become: yes
  tasks:
    - name: Create users
      include_role:
        name: create-users-role
  vars:
    users_file: users-role.yml
    passwords_file: users-passwords-role.yml
```

Мы внесли все необходимые изменения, для того чтобы реализовать роль. Теперь можно удалить все пустые или неиспользуемые подкаталоги в каталоге роли `create-users-role`.

Теперь давайте проверим нашу роль.

Как протестировать роль

Чтобы протестировать роль, мы воспользуемся сценариями в каталоге `tests` и запустим их с помощью таких команд:

```
ansible-playbook create-users/tests/test.yml
ansible-playbook create-users/tests/test2.yml
```

Убедитесь, что обе команды завершились успешно.

Итак, мы представили вашему вниманию обзор ролей, которые являются мощной функцией Ansible и позволяют современным системным администраторам и специалистам по DevOps оперативно переходить от концепции к реализации и быстрее развертывать повседневные рабочие процессы управления конфигурацией.

Итоги

В этой главе мы рассмотрели основные понятия Ansible, хотя из-за ограниченного объема книги не смогли охватить весь богатый ассортимент возможностей этой платформы. Тем не менее мы дали общее представление о ней, начиная с архитектурных принципов Ansible и заканчивая тем, как настраивать среду и как работать со специальными командами и сценариями. В практических упражнениях вы настроили среду Ansible с несколькими управляемыми узлами и одним управляющим узлом, в общих чертах имитируя реальное развертывание. Вы также познакомились с тем, как составлять команды и писать сценарии Ansible для типичных задач управления конфигурацией. Большинство команд и сценариев, которые представлены в этой главе, очень похожи на повседневные административные операции.

Независимо от того, являетесь ли вы системным администратором или инженером DevOps, опытный ли вы специалист или только собираетесь им стать, мы

надеемся, что эта глава помогла вам по-новому взглянуть на повседневные задачи администрирования Linux и процессы автоматизации. Инструменты и методы, которые вы здесь изучили, послужат хорошим фундаментом для того, чтобы писать сценарии и автоматизировать ваши каждодневные административные процедуры.

Эти же заключительные мысли относятся и к книге в целом. Вы проделали большой путь и освоили немало типичных задач администрирования Linux как в локальных, так и в облачных средах.

Мы надеемся, что вам понравилось наше совместное путешествие.

Вопросы

Давайте подведем итог основным понятиям, о которых вы узнали в этой главе:

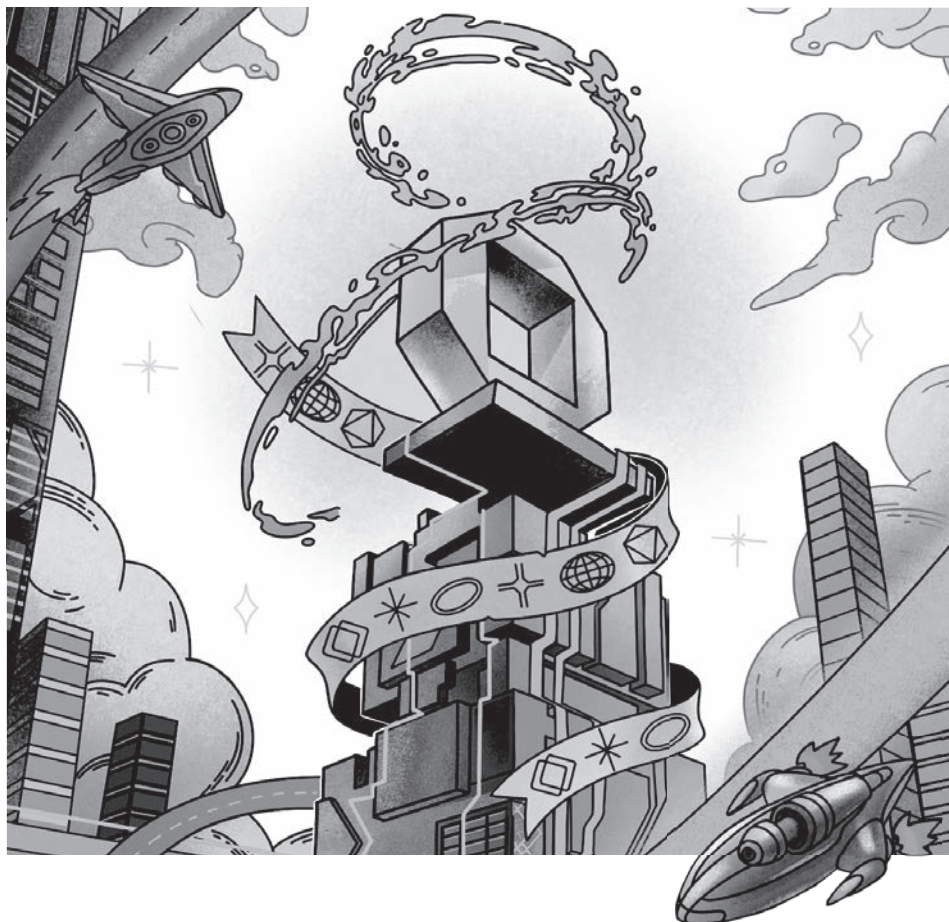
1. Что такое идемпотентные операции (команды) в Ansible?
2. Как настроить аутентификацию без пароля на управляемых узлах?
3. Как выглядит разовая команда, которая позволяет проверить связь со всеми управляемыми узлами?
4. Перечислите несколько модулей Ansible. Попробуйте придумать сценарий управления конфигурацией, в котором можно использовать каждый модуль.
5. Напишите простой сценарий, который отслеживает, сколько памяти доступно на узлах, и уведомляет вас, если ее объем меньше некоторого заданного порога.

Дополнительная литература

Вот несколько полезных ресурсов, которые помогут вам лучше познакомиться с Ansible.

Документация по Ansible: docs.ansible.com:

- Примеры использования Ansible от Red Hat: redhat.com/technologies/management/ansible/use-cases.
- Джеймс Спурин (James Spurin). «Dive into Ansible — From Beginner to Expert», видеокурс.
- Дэниел Ох, Джеймс Фриман, Фабио Алессандро Локати (Daniel Oh, James Freeman, Fabio Alessandro Locati). «Practical Ansible: Learn how to automate infrastructure, manage configuration, and deploy applications», Second Edition.



Эту книгу мы выпустили вместе с Orion soft

Orion soft создает ПО для ИТ-инфраструктуры Enterprise-бизнеса. Главные принципы разработки, которым отвечает каждый продукт компании, — стабильность, простота использования и безопасность.

В продуктовую экосистему Orion soft входит zVirt — платформа виртуализации серверов, дисков и сетей с одной из крупнейших баз инсталляций в России.

Orion soft стремится постоянно развивать как свои компетенции, так и российское ИТ-сообщество, собирая и распространяя лучшие современные практики разработки.