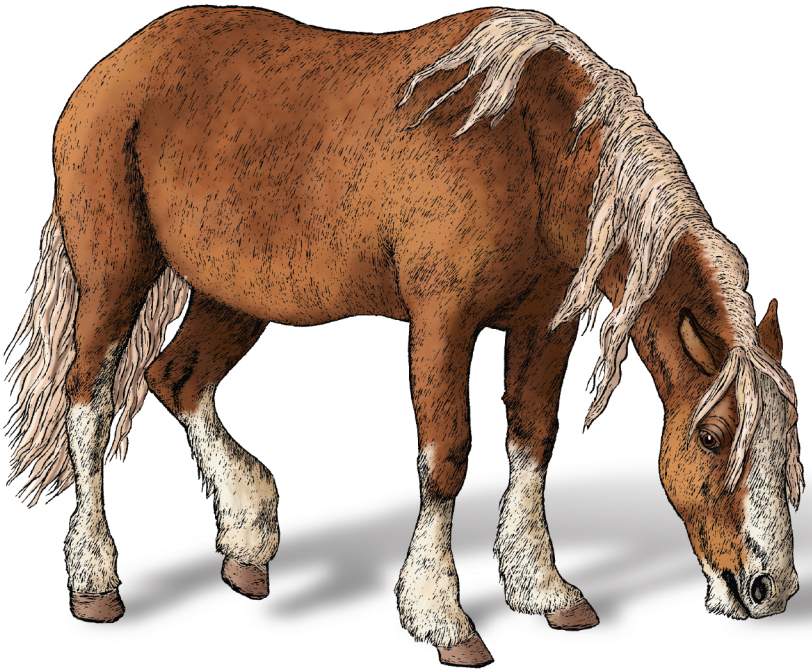
O'REILLY®

Четвертое издание

Linux

карманный справочник



SPRINT book

Дэниел Джей Барретт

4TH EDITION

Linux Pocket Guide

Daniel J. Barrett

Beijing • Boston • Farnham • Sebastopol • Tokyo

O'REILLY®

ЧЕТВЕРТОЕ ИЗДАНИЕ

Linux. Карманный справочник

Дэниел Джей Барретт

SPRINT book 2025

ББК 32.973.2-018.2
УДК 004.451
Б25

Барретт Дэниел Джей

Б25 Linux. Карманный справочник. 4-е изд. — Астана: «Спринт Бук», 2025. — 320 с.: ил.

ISBN 978-601-08-4416-2

Книга идеально подойдет всем, кто каждый день пользуется Linux. В обновленном издании описывается более 200 команд Linux, в том числе новые команды управления файлами, пакетами и версиями исходного кода, преобразования форматов файлов и многие другие.

Автор книги Дэниел Барретт приводит наиболее полезные команды Linux, сгруппированные по функциональности. Это практическое издание станет идеальным справочником по Linux и для новичков, и для опытных пользователей.

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

ББК 32.973.2-018.2
УДК 004.451

ISBN 978-1098157968 англ.

Authorized Russian translation of the English edition of Linux Pocket Guide, 4E
ISBN 9781098157968 © 2024 Daniel J. Barrett
This translation is published and sold by permission of O'Reilly Media, Inc., which owns or controls all rights to publish and sell the same.

ISBN 978-601-08-4416-2

Оглавление

Отзывы о книге

«Linux. Карманный справочник» — книга, которую должен прочитать каждый пользователь Linux. Это коллекция избранных заметок, которые я время от времени пересматриваю. Книга написана простым и легким для понимания языком, так что у вас не возникнет никаких вопросов!

Абхишек Пракаш, соучредитель It's FOSS

Моя самая любимая особенность среды Linux — это набор небольших утилит для решения проблем. В этой книге вся информация представлена в виде справочника. Даже опытные пользователи смогут освежить свои знания и откроют новые грани и невероятные возможности привычных инструментов.

Джесс Майлз, DevOps-инженер, TriumphPay

Очень удобный справочник! Автор проделал удивительную работу: он собрал полную информацию по Linux и уместил ее в этой маленькой книге!

Джерод Санто, changelog.org

Коротко о главном

Добро пожаловать в Linux! Если вы только начинаете свой путь, то эта книга послужит вам лаконичным введением и руководством по распространенным командам. Если вы уже опытный пользователь, то можете пропустить вводную часть.

О чем эта книга

Эта книга — краткое руководство, а не *учебное пособие*. Я рассказываю только о важных и полезных аспектах Linux, чтобы вы могли работать продуктивно. Я рассматриваю лишь некоторые команды и параметры (прошу прощения, если не упомянул вашу любимую команду) и не вдаюсь в подробности внутреннего устройства операционной системы. Мой девиз: «Коротко и по существу».

Я сосредоточусь на *командах*, которые вводятся в оболочку командной строки и передаются системе Linux. Вот пример команды, подсчитывающей строки текста в файле `myfile`:

```
wc -l myfile
```

В книге рассматриваются важные для большинства пользователей команды Linux, например `ls` (вывод содержимого каталога), `grep` (поиск текста), `mplayer` (воспроизведение аудио- и видеофайлов) и `df` (выводит информацию о свободном месте на диске). Я вскользь упоминаю графические среды, например GNOME и KDE Plasma, потому что о них можно написать отдельные *краткие руководства*.

Материал в книге упорядочен по функциям. Например, чтобы посмотреть содержимое файла, я собрал в одном месте множество команд просмотра файлов: `cat` — для маленьких текстовых файлов, `less` — для больших текстовых файлов,

`od` — для двоичных и т. д. Затем рассматриваю каждую команду, рассказываю о ее использовании и параметрах.

Я подразумеваю, что у вас есть доступ к системе Linux. Если доступа нет, то попробовать поработать в Linux можно на большинстве компьютеров. Вам просто нужно развернуть дистрибутив Linux на USB-носителе. Примерами являются Ubuntu (https://oreil.ly/ralRq), Fedora (https://oreil.ly/Y3QGZ) и KNOPPIX (https://oreil.ly/Byqeu).

Что нового в четвертом издании?

Новые команды

Я добавил 50 новых команд, например `git` и `svn` для управления версиями, `split` и `column` для работы с текстом, `pandoc` и `ffmpeg` для конвертирования файлов, `snap` и `flatpak` для управления пакетами, `mdadm`, `lvcreate` и `zfs` для управления хранилищами, `gpg` для шифрования и т. д.

Четкая организация

Книга разделена на главы, посвященные концепциям, файлам, основам системного администрирования, работе в Сети и другим темам.

Прощайте, древние команды

Некоторые команды из предыдущих изданий устарели, например `write` и `finger`, или перестали работать, как `ftp`. Я заменил их более актуальными и современными.

Синтаксис команд

Здесь при рассмотрении команд используется следующая структура: сначала указывается стандартный синтаксис команды. На рис. В.1 показан синтаксис команды `ls`, которая выводит имена и атрибуты файлов. В упрощенной форме синтаксиса отражено распространенное применение команды:

```
ls [параметр(ы)] [файл(ы)]
```

Это означает, что если вы введете `ls`, то сначала должны указать *параметры*, а затем имена *файлов*. При наборе команд квадратные скобки `[]` набирать не нужно — они лишь указывают, что содержимое в них вводить не обязательно. Выделенные курсивом слова означают, что вместо них вам нужно привести собственные значения, например имена реальных файлов. Если между параметрами или аргументами есть вертикальная черта:

```
(файл | каталог)
```

это означает выбор — в качестве аргумента вы можете указать либо имя *файла*, либо имя *каталога*.

Рис. B.1. Стандартный синтаксис команды

В стандартном синтаксисе, приведенном на рис. B.1, шесть свойств команды показаны разным шрифтом. Черный означает, что свойство поддерживается, а серый — не поддерживается.

stdin
: По умолчанию программа считывает данные со стандартного ввода (клавиатуры). См. раздел «Ввод, вывод и перенаправление» главы 1.

stdout
: По умолчанию команда передает данные на стандартный вывод (экран). См. раздел «Ввод, вывод и перенаправление» на главы 1.

`-file`
: Аргумент в виде одинарного дефиса (`-`) указывает, что нужно считывать данные со стандартного ввода, а не из

файла на диске. Аналогично, если дефис используется в качестве результирующего вывода, то команда запишет данные в стандартный вывод. Так, показанная далее команда `wc` считывает файлы `myfile` и `myfile2`, затем стандартный ввод, затем `myfile3`:

```
wc myfile myfile2 - myfile3
```

`--opt`

Параметр в виде двойного дефиса (`--`) означает «конец параметров» (options): все строки, указанные далее, не относятся к параметрам. Работая с файлами, имена которых начинаются с дефиса, необходимо указывать двойной дефис, чтобы файл не был ошибочно принят за параметр. Например, если у вас есть файл `-dashfile`, то команда `wc –dashfile` не будет выполнена, так как параметр `–dashfile` будет считаться некорректным. Корректная команда будет выглядеть вот так: `ws -- -dashfile`. Если команда не поддерживает двойные дефисы, то вы можете добавить к имени файла путь к текущему каталогу (`./`), чтобы дефис не был первым символом:

```
wc ./-dashfile
```

`--help`

`--help` выводит справочную информацию о команде, а затем завершает работу.

`--version`

`--version` выводит информацию о версии команды, а затем завершает работу.

Команды, подсказки и вывод информации

Командная строка Linux, или *оболочка*, в режиме ожидания команды выводит специальный символ — *приглашение*. В этой книге приглашение — направленная вправо стрелка:

```
→
```

Приглашения бывают разных форм и размеров, это зависит от настроек оболочки. В качестве приглашения может использоваться знак доллара (`$`), сочетание имени вашего компьютера с именем пользователя либо несколькими символами (`myhost:~smith$`) или любой другой символ. Все приглашения означают одно и то же: оболочка готова принять следующую команду.

В книге я буду приводить в качестве примера командные строки. Одни из них должны набирать пользователи, другие — нет (например, приглашения или вывод команды). Полужирным шрифтом я выделяю фрагменты, которые вам нужно набирать самостоятельно. Курсивом выделены комментарии с объяснением того, что происходит:

```
→ wc -l myfile          Приглашение для ввода команды в командной строке
18 myfile               Результат выполнения команды
```

Ваш верный друг — команда echo

Во многих примерах я вывожу информацию на экран с помощью команды `echo`. Подробнее я описываю ее в разделе «Вывод на экран» главы 6. Это одна из простейших команд, она выводит свои аргументы на стандартный вывод после того, как аргументы были обработаны оболочкой:

```
→ echo My dog has fleas
My dog has fleas
→ echo My name is $USER          Переменная оболочки USER
My name is smith
```

Длинные командные строки

Когда длина команды превышает ширину страницы книги, приходится разделять ее на несколько строк. Строка должна оканчиваться обратным слешем (\), означающим «продолжение на следующей строке»:

```
→ echo This is a long command that does not fit on \
  one line
This is a long command that does not fit on one line
```

Разделяйте длинные команды с помощью слеша или просто вводите всю команду в одну строку без использования специальных символов.

Сочетания клавиш

Я использую определенные символы для обозначения клавиш клавиатуры. Карет (^) означает «удерживание нажатой клавиши управления» — обычно это клавиша Ctrl. Например, ^D (Ctrl+D) означает: «Нажав и удерживая клавишу Ctrl, нажмите клавишу D». Когда я пишу Esc, то хочу сказать: «Нажмите и отпустите клавишу Escape». Такие клавиши, как Enter и Пробел, пояснений не требуют.

Загрузка файлов примеров

Я собрал целую коллекцию файлов, которые помогут вам практиковаться в работе с Linux. Можете скачать их и использовать на любом устройстве под управлением операционной системы Linux. Чтобы загрузить файлы в первый раз, выполните следующие команды[1] (обратите внимание на то, что параметр `-O` содержит прописную букву «O», а не ноль):

```
→ cd
→ curl -O https://linuxpocketguide.com/LPG4.tar.gz
→ tar -xf LPG4.tar.gz
```

Эти команды создают каталог `linuxpocketguide` в домашнем каталоге. Теперь вам нужно перейти в этот каталог:

```
→ cd ~/linuxpocketguide
```

[1] Если у вас есть опыт работы с git и GitHub, загрузите файлы, перейдя по ссылке (https://resources.oreilly.com/oreillymedia/linux_pocket_guide_4), и пропустите мои инструкции. Если вы клонируете репозиторий и хотите вернуть файлы в изначальное состояние, не запускайте скрипт reset-lpg — вместо этого выполните команду git reset --hard.

Выполняйте команды по мере чтения книги. Результат команды должен совпадать с показанным в книге, за исключением некоторых деталей, например дат и имен пользователей.

Чтобы снова загрузить и установить файлы (например, если вы их изменили), просто запустите сценарий `reset-lpg`:

```
→ cd ~/linuxpocketguide
→ bash reset-lpg
```

Если вы поместили файлы в другой каталог, то учтите изменения в команде `reset-lpg`. Следующая команда создает или обновляет каталог `/tmp/practice/linuxpocketguide`:

```
→ bash reset-lpg /tmp/practice
```

Условные обозначения

В этой книге используются следующие шрифтовые выделения:

Курсив
: Курсивом выделяются новые термины.

Рубленый шрифт
: Им обозначены URL-адреса, названия клавиш и адреса электронной почты.

`Моноширинный шрифт`
: Применяется для оформления листингов программ, а также внутри обычного текста для обозначения отдельных элементов кода, например имен переменных и функций, баз и типов данных, переменных окружения, состояния или ключевых слов, имен и расширений файлов.

`Полужирный моноширинный шрифт`
: Применяется для команд и прочего текста, который должен набирать пользователь.

Курсивный моноширинный шрифт

Применяется для оформления текста, который нужно заменить на значения пользователя или определяемые контекстом.

СОВЕТ

Данный элемент означает совет или предложение.

ПРИМЕЧАНИЕ

Данный элемент означает примечание.

ВНИМАНИЕ!

Данный элемент означает предупреждение.

Благодарности

Я признателен моим читателям, которые купили первые три издания книги за последние 20(!) лет, — только благодаря вам я смог выпустить четвертое издание. Выражаю искреннюю благодарность моему редактору Вирджинии Уилсон, рецензенту издательства Джону Девинсу, команде издательства O'Reilly, моим замечательным техническим редакторам Абхишеку Пракашу, Дэну Риттеру, Дорону Бейт-Халамхи, Итану Шварцу и Джесс Малес, а также Мэгги Джонсон из Google, Кери и Лесли Минниар из Alucard Music. Хочу поблагодарить и свою замечательную семью — Лизу, Софию, Кея и Луну.

От издательства

Ваши замечания, предложения, вопросы отправляйте по адресу

comp@sprintbook.kz

(издательство «SprintBook», компьютерная редакция).

Мы будем рады узнать ваше мнение!

ГЛАВА 1

Основные концепции

Что такое Linux

Linux — это бесплатная операционная система (ОС) с открытым исходным кодом, являющаяся альтернативой Microsoft Windows и Apple macOS. Под управлением операционной системы Linux действует большинство серверов в интернете. Linux работает в фоновом режиме на всех смартфонах Android и ноутбуках Chromebook, а также на миллионах подключенных к сети устройств, например маршрутизаторах, брандмауэрах и даже роботизированных системах доения коров (я серьезно). ОС также отлично запускается на ноутбуках и стационарных компьютерах.

Linux формируют четыре компонента (рис. 1.1).

Ядро

Низкоуровневое программное обеспечение. Управляет аппаратным обеспечением и основными функциями, например планированием процессов или работой в сети. Обычно пользователи не взаимодействуют с ядром напрямую.

Программное обеспечение

Тысячи программ для работы с файлами, редактирования текста, разработки ПО, просмотра веб-страниц, аудио, видео, шифрования, математических вычислений... Все эти программы взаимодействуют с ядром. Программы, запускаемые в оболочке командной строки, называются *командами*.

Рис. 1.1. Четыре основных компонента Linux. Низкоуровневые функции ядра вызываются программами, которые вызываются в оболочке. В свою очередь, оболочка может быть запущена из графической среды Рабочего стола

Оболочка

Используется для выполнения команд и отображения результатов. В Linux есть множество командных оболочек с различными возможностями. В этой книге рассматривается оболочка bash, которая чаще всего применяется по умолчанию в учетных записях пользователей. Остальные оболочки — dash, fish, ksh (оболочка Korn), tcsh (оболочка TC), zsh (оболочка Z), а также busybox. Все они имеют схожие функции, но их применение может различаться.

Графическая среда Рабочего стола (опционально)

Пользовательский интерфейс (UI) с окнами, меню, значками, поддержкой мыши и другими привычными элементами графического интерфейса пользователя (GUI). Самыми популярными средами являются GNOME и KDE Plasma. Большинство приложений,

созданных для GNOME, могут работать в KDE, и наоборот[1].

Эта книга фокусируется на командной части Linux, а именно на программном обеспечении и оболочке. В Windows и macOS тоже есть интерфейс командной строки (`cmd` и `powershell` в Windows, Terminal в macOS), хотя большинство пользователей применяют графический интерфейс и никогда не видят командную строку. В Linux оболочка играет решающее значение: если вы используете ее без оболочки, то многое упускаете.

Linux очень легко модифицировать, так что вы можете найти сотни вариантов оболочки, подходящих для тех или иных нужд. Каждая версия называется *дистрибутивом*. Все дистрибутивы имеют общие компоненты, но могут выглядеть по-разному и включать в себя разное ПО. Самыми популярными дистрибутивами считаются Mint, Ubuntu, Manjaro, Arch, Gentoo, Red Hat и OpenSUSE. Материал данной книги можно применить к любому из них.

Запуск оболочки

Как запускается оболочка? Обычно Linux запускает ее автоматически, например, когда вы входите в систему с помощью `ssh` или аналогичной программы. Первое, что вы увидите, — приглашение оболочки, ожидающее команды.

Иногда оболочку приходится запускать вручную. Это типично для Рабочих столов со множеством значков и меню, в котором нет оболочки. Тогда вам понадобится приложение с графическим интерфейсом — *терминал*, или *терминальная программа*. С помощью терминала можно запустить оболочку в окне. Во врезке «Оболочка и терминал» далее я объясняю разницу между оболочками и терминалами.

---

[1] GNOME, KDE и другие графические среды используют общий оконный интерфейс, например X Window System или Wayland. Чтобы узнать, какая система у вас, выполните команду $XDG_SESSION_TYPE.

В каждом дистрибутиве с графической средой Рабочего стола есть хотя бы одна терминальная программа. Чтобы запустить ее, поищите приложение, значок или пункт меню с именем Terminal, Konsole, xterm, gnome-terminal, uxterm или подобным. Запустите это приложение — откроется терминал. В некоторых средах можно открыть его нажатием клавиш Ctrl+Alt+T (нажав и удерживая клавиши Ctrl и Alt, нажмите клавишу T).

Оболочка и терминал

Оболочка — это интерфейс командной строки для запуска команд путем ввода обычного текста. В командной строке отображается приглашение:

```
→
```

Терминал — это программа, открывающая новое окно и запускающая оболочку. Я показал терминал на рис. 1.2. Он добавляет меню, полосу прокрутки, функцию копирования и вставки, и также другие функции графического интерфейса, поддерживающие оболочку.

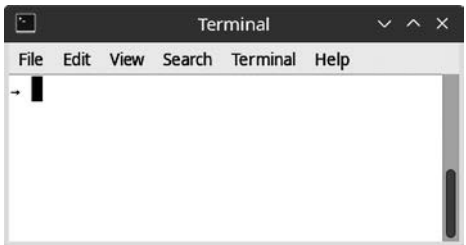


Рис. 1.2. Терминал открывает окно, в котором запускается оболочка

Активация командной строки

Чтобы показать работу Linux, попробуем выполнить в оболочке десять простых команд. Набирайте их *точно* как в книге, включая прописные и строчные буквы, пробелы

и символы после приглашения. В конце каждой команды нажмите клавишу Enter[1].

Отобразите календарь на ноябрь 2023 года:

```
→ cal nov 2023
    November 2023
Su Mo Tu We Th Fr Sa
          1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30
```

Отобразим содержимое каталога `/bin`, в котором содержится множество команд:

```
→ ls /bin
bash       less          rm
bunzip2    lessecho      rmdir
busybox    lessfile      rnano
⋮
```

Подсчитайте количество видимых элементов в домашнем каталоге (здесь я использую переменную `HOME`, о которой расскажу позже):

```
→ ls $HOME | wc -l
8
```

Ваш результат может отличаться

Проверьте, сколько места занято на определенном разделе жесткого диска:

```
→ df -h /
Filesystem   Size   Used Avail Use% Mounted on
/dev/sdb1     78G    30G   48G  61% /
```

Откройте список процессов, запущенных на вашем компьютере (для выхода нажмите клавишу Q):

```
→ top -d1
```

---

[1] Если видите сообщение об ошибке «Команда не найдена», не волнуйтесь — возможно, команда не установлена в вашей системе. См. раздел «Установка ПО» главы 3.

Выведите содержимое файла `/etc/hosts`, в котором указаны имена и адреса компьютеров (на принтер по умолчанию, если он подключен):

```
→ lpr /etc/hosts
```

Узнайте, когда вы вошли в систему:

```
→ last -1 $USER
smith  pts/7 :0  Tue Nov 10 20:12  still logged in
```

Скачайте файл `sample.pdf` с сайта книги в текущий каталог без веб-браузера:

```
→ curl -O https://linuxpocketguide.com/sample.pdf
```

Посмотрите, кому принадлежит доменное имя `oreilly.com` (для перехода на страницу нажмите клавишу Пробел, а для выхода — клавишу Q):

```
→ whois oreilly.com | less
Domain Name: OREILLY.COM
Registrar: GODADDY.COM, LLC
⋮
```

Наконец, очистите терминал:

```
→ clear
```

Поздравляю, теперь вы пользователь Linux!

Структура команд

Обычно команда Linux состоит из *имени программы*, *параметров* и *аргументов*:

```
wc -l файл
```

Имя программы (`wc`, сокращение от *word count* — подсчет слов) относится к программе на диске, которую оболочка находит и запускает. *Параметры*, обычно начинающиеся с дефиса, влияют на поведение программы. В предыдущей команде параметр `-l` указывает `wc` считать строки, а не слова. Аргумент *`файл`* определяет файл, с которым будет работать команда `wc`.

Команды могут иметь несколько параметров и аргументов. Параметры могут быть указаны по отдельности или с общим дефисом:

```
wc -l -w файл        Два отдельных параметра
wc -lw файл          Объединенные параметры, то же самое, что -l -w
```

ВНИМАНИЕ!

Некоторые программы не распознают объединенные параметры.

Вы также можете указать несколько аргументов:

```
wc -l файл1 файл2    Подсчет строк в двух файлах
```

Параметры не стандартизированы. Они могут состоять из одного дефиса и символа (например, `-l`), двух дефисов и слова (`--lines`) и т. п. Один и тот же параметр может иметь разное значение для разных программ: в команде `wc -l` параметр `-l` означает «строки текста», а в команде `ls -l` — «длинный вывод». Разные программы могут использовать разные параметры для обозначения одного и того же процесса. Так, параметры `-q` и `-s` означают «тихий запуск». После некоторых параметров следует указать значение, например `-s 10`. Пробел можно не вставлять (`-s10`).

Обычно в качестве аргументов выступают имена файлов для ввода *(источник)* или вывода *(назначение)*, но вместо них могут использоваться и имена каталогов, пользователей, хостов, IP-адреса, выражения и прочие данные.

Команда, состоящая из одной программы с параметрами и аргументами, называется *простой*. Далее показана простая команда, которая выводит список пользователей, вошедших в систему на сервере Linux[1]:

[1] Пользователь silver, указанный в списке дважды, запустил две интерактивные оболочки одновременно.

```
→ who
silver          :0      Sep 23 20:44
byrnes       pts/0      Sep 15 13:51
barrett      pts/1      Sep 22 21:15
silver       pts/2      Sep 22 21:18
```

Команда может вызывать несколько программ одновременно и даже настраивать их на взаимодействие друг с другом. Ниже приведен пример команды, передающей вывод команды `who` команде `wc`, которая подсчитывает строки текста. Результатом будет количество строк в выводе команды `who`:

```
→ who | wc -l
4
```

Вертикальная полоса, называющаяся *пайпом*, обеспечивает перенаправление данных между `who` и `wc`. Опытные пользователи Linux постоянно применяют подобные объединенные команды, называемые *конвейерами*.

Команды могут включать такие элементы языка программирования, как переменные, условия и циклы, о которых я рассказываю в разделе «Программирование с помощью сценариев командной оболочки» главы 6. Например, команда может гласить: «Запустить программу, записать вывод в указанный файл и, если возникнут ошибки, вывести сообщение с ошибкой».

Пользователи и суперпользователи

Linux — это многопользовательская ОС: несколько человек могут одновременно запустить несколько программ на одном компьютере. У каждого из них есть *имя пользователя*, например `smith` или `funkydance`. У каждого есть отдельное рабочее пространство (см. раздел «Домашние каталоги» далее в этой главе), чтобы не мешать друг другу.

Специальные пользователи с именем `root` — *суперпользователи* или администраторы — обладают всеми правами в системе. Суперпользователь может создавать, изменять и удалять любые файлы и запускать любые программы. Обычные пользователи ограничены в правах: хоть они

и могут запустить большинство программ, они не могут вмешиваться в работу других пользователей.

Для выполнения некоторых команд из этой книги требуются права суперпользователя. Я добавляю к таким командам слово `sudo`:

```
→ sudo далее указывается команда, требующая прав
суперпользователя
```

ВНИМАНИЕ!

С помощью команд sudo можно вывести из строя вашу систему Linux!

Я подробно рассказываю о привилегиях `sudo` в разделе «Привилегии суперпользователя» главы 3, а пока что достаточно знать, что `sudo` дает вам права суперпользователя. Иногда команда может запросить ваш пароль — например, если вы захотите подсчитать строки в защищенном файле `/etc/shadow`. Чтобы сделать это, выполните следующую команду:

```
→ wc -l /etc/shadow              Ошибка!
wc: /etc/shadow: Permission denied
→ sudo wc -l /etc/shadow         Запуск команды от лица суперпользователя
[sudo] password: xxxxxxxx
51 /etc/shadow                   Сработало!
```

Файловая система

Чтобы использовать систему Linux, вы должны научиться работать с файлами и каталогами Linux (они же папки), которые и представляют собой *файловую систему*. В графической среде Рабочего стола вы видите файлы и каталоги. В интерфейсе командной строки можно найти те же файлы и каталоги, однако выполнить это сложнее, поэтому нужно помнить, в каком каталоге вы находитесь и как он связан с другими каталогами. Вы будете использовать такие команды, как `cd` (сменить каталог), для перемещения

между каталогами и `pwd` (вывод рабочего каталога), чтобы отслеживать свое положение в файловой системе.

Разберемся с терминологией. Файлы в Linux организованы в *каталоги*. Последние формируют иерархическую структуру, или *дерево* (рис. 1.3). Один каталог может включать в себя другие каталоги, называемые *подкаталогами*, которые также могут включать в себя другие файлы и подкаталоги. Самый первый, верхний каталог называется *корневым* и обозначается слешем (`/`)[1].

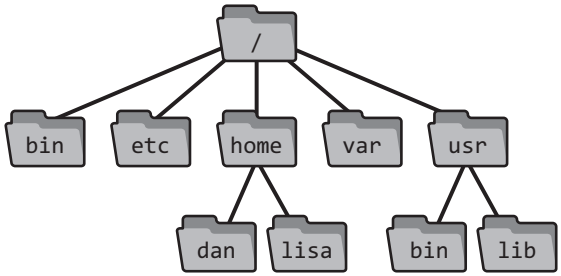


Рис. 1.3. Файловая система Linux (фрагмент). Корневой каталог располагается вверху. Например, путь к папке dan выглядит как /home/dan

Linux ссылается на файлы и каталоги согласно синтаксису «имен и слешей», называемому *путем*. Например, путь

```
/один/два/три/четыре
```

ссылается на корневой каталог `/`, содержащий каталог *один*, который содержит каталог *два*, который содержит каталог *три*, который содержит конечный файл или каталог *четыре*. Любой путь, начинающийся со слеша и идущий от корня, называется *абсолютным (полным)* путем.

Пути не обязательно должны быть абсолютными — они могут быть относительными и вести к какому-либо каталогу,

[1] В Linux все файлы и каталоги располагаются в соответствии с иерархией, начиная с корневого каталога. В этом состоит отличие от Windows, где доступ к различным устройствам осуществляется с помощью букв логических разделов.

отличному от корневого. На рис. 1.3 показаны два разных каталога с именем `bin`, абсолютные пути к которым — `/bin` и `/usr/bin`. Если вы будете просто ссылаться на каталог `bin`, то будет непонятно, какой именно вы имеете в виду. Требуется более подробный контекст. Любой путь, не начинающийся со слеша (как в случае с `bin`), называется *относительным*.

Чтобы правильно использовать относительные пути, нужно знать, где вы находитесь в файловой системе Linux. Ваша локация называется *текущим каталогом* (иногда активным каталогом или текущим рабочим каталогом).

У каждой оболочки есть текущий каталог, и когда вы запускаете команды в этой оболочке, они выполняются в текущем каталоге. Например, если в вашей оболочке открыт каталог `/usr`, а вы ссылаетесь на относительный путь `bin`, то полный путь будет выглядеть так: `/usr/bin`. Или если ваш текущий каталог — `один/два/три`, а относительный путь — `a/b/c`, то абсолютный путь будет `/один/два/три/a/b/c`.

Обычно *текущий* каталог оболочки обозначается `.` (точкой), а *родительский* каталог — `..` (двумя точками). Следовательно, если текущий каталог оболочки — `один/два/три`, то `.` относится к этому каталогу, а `..` относится к `один/два`.

Чтобы перейти из одного каталога в другой, используйте команду `cd`, которая меняет текущий каталог вашей оболочки:

```
→ cd /usr/local/bin      Переход в каталог /usr/local/bin
```

В команде `cd` указан абсолютный путь. С помощью `cd` можно выполнять и относительные перемещения:

```
→ cd d                   Переход в подкаталог текущего каталога
→ cd ../mydir            Переход в родительский каталог, затем в каталог mydir
```

Имена файлов и каталогов могут содержать большинство символов: прописные и строчные буквы[1], цифры, точки,

[1] Имена файлов в Linux чувствительны к регистру, поэтому следует учитывать прописные и строчные буквы.

дефис, знаки подчеркивания и т. д., за исключением символа слеша (/), который используется для разделения каталогов. Для большей эффективности избегайте пробелов, звездочек, знака доллара, круглых скобок и других символов, имеющих специальное значение для оболочки. Такие символы в именах файлов требуют специального экранирования (см. раздел «Заключение в кавычки» далее в этой главе).

Домашние каталоги

В большинстве своем персональные файлы обычных пользователей хранятся в каталоге `/home`, а суперпользователей — в `/root`. Домашний каталог чаще всего обозначается как `/home/`*`ваше_имя_пользователя`* (`/home/smith`, `/home/rightman` и т. д.). Существует несколько способов перейти или обратиться к домашнему каталогу:

`cd`

Без аргументов команда возвращает вас (то есть устанавливает текущий каталог оболочки) в домашний каталог.

`HOME`

Переменная `HOME` (см. раздел «Переменные оболочки» далее в этой главе) содержит имя вашего домашнего каталога:

```
→ echo $HOME                          Вывод имени каталога
/home/smith
→ cd $HOME/linuxpocketguide           Переход в подкаталог
```

`~`

Если вместо имени каталога указать символ «тильда», то он будет интерпретирован оболочкой как имя вашего домашнего каталога.

```
→ echo ~                              Вывод имени каталога
/home/smith
→ cd ~/linuxpocketguide               Переход в подкаталог
```

Если за тильдой указано имя пользователя (например, `~fred`), то оболочка интерпретирует эту строку в виде домашнего каталога пользователя:

```
→ cd ~fred          Переход в домашний каталог Фреда, если он существует
→ pwd               Команда "вывода рабочего каталога"
/home/fred
```

Системные каталоги

В системе Linux есть десятки тысяч системных каталогов. Они содержат файлы операционной системы, приложения, документацию и практически все, кроме персональных файлов пользователей, которые обычно находятся в каталоге `/home`.

Если вы не системный администратор, то редко будете сталкиваться с системными каталогами. Но если вы овладеете некоторыми знаниями, то сможете догадаться о значении большинства системных каталогов. Очень часто их имена состоят из трех частей (рис. 1.4).

Рис. 1.4. Область, категория и приложение каталога

Путь каталога, часть 1. Область

Область пути каталога описывает назначение всего дерева каталогов. Некоторые из них являются общими:

`/`	(корень) Системные файлы, относящиеся к вашему дистрибутиву
`/usr`	(пользователь) Дополнительные системные файлы, относящиеся к вашему дистрибутиву
`/usr/local`	(локальный пользователь) Системные файлы, не относящиеся к вашему дистрибутиву. Они могут быть уникальными для вашей локальной сети Linux или компьютера

На практике четкого различия между / и /usr нет, но / считается более «низкоуровневым» подходом в отношении ОС.

Путь каталога, часть 2. Категория

Категория пути каталога (см. рис. 1.4) описывает типы файлов, находящихся в каталоге. Например, если указана категория lib, то он содержит библиотечные файлы для программирования. Если указана категория bin, то в каталоге находятся двоичные файлы — исполняемые программы.

Если перед категорией bin указана область, у нас получаются такие пути, как /bin, /usr/bin и /usr/local/bin. Основные системные программы дистрибутива, например ls и cat, обычно находятся в /bin, а другие системные программы — в /usr/bin[1]. Каталог /usr/local/bin включает в себя локально установленные программы, не входящие в состав вашего дистрибутива. Это типичные случаи, а не жесткие правила.

Давайте рассмотрим некоторые распространенные категории.

Категории для программ

bin	Программы (обычно исполняемые файлы)
sbin	Программы (обычно исполняемые файлы) для суперпользователей
lib	Библиотеки кода, применяемые программами

Категории для документации

doc	Документация
info	Файлы документации для встроенной системы eMac
man	Файлы документации (man-страницы), отображаемые командой man; файлы часто сжимаются и содержат инструкции, интерпретируемые этой командой
share	Файлы, специфичные для программы, например инструкции по установке

[1] Сейчас между некоторыми каталогами нет таких различий. Например, в дистрибутиве Fedora каталог /bin символично ссылается на /usr/bin.

Категории для конфигураций

`etc`	Файлы конфигурации системы и прочие похожие
`init.d`	Файлы конфигурации для загрузки Linux
`rc.d`	Файлы конфигурации для загрузки Linux, а также `rc1.d`, `rc2.d`…

Категории для программирования

`include`	Заголовочные файлы для программирования
`src`	Исходный код программ

Категории для веб-файлов

`cgi-bin`	Сценарии/программы, запускаемые на веб-страницах
`html`	Веб-страницы
`public_html`	Веб-страницы, находящиеся в домашних каталогах пользователей
`www`	Веб-страницы

Категории для отображения

`fonts`	Шрифты
`X11`	Системные файлы X-Window

Категории для устройств

`dev`	Файлы устройств для взаимодействия с дисками и другим оборудованием
`media`, `mnt`	Точки монтирования — каталоги, обеспечивающие доступ к дискам

Категории для файлов среды выполнения

`var`	Файлы, связанные с состоянием компьютера, часто обновляются
`lock`	Программы создают блокируемые файлы для обозначения статуса «программа уже запущена». Такой файл может помешать другой программе или экземпляру той же команды выполнить действие
`log`	Файлы журналов, в которых фиксируются важные системные события и сообщения об ошибках, предупреждения и информационные сообщения
`mail`	Почтовые ящики для входящей электронной почты
`run`	PID-файлы, содержащие идентификаторы запущенных процессов. К ним часто обращаются для отслеживания или удаления определенных процессов
`spool`	Файлы в очереди или пути, например исходящая электронная почта, задания на печать или запланированные задачи
`tmp`	Временное хранилище для применения программами и/или пользователями

Путь каталога, часть 3. Приложение

Эта часть пути каталога (см. рис. 1.4) обычно является именем программы. Например, путь каталога `/etc/systemd` указывает на корневую область `/`, категорию `etc` (конфигурационные файлы) и приложение `system`. Так как `system` — это служба для настройки Linux-систем, можно предположить, что каталог `/etc/system` содержит конфигурационные файлы для нее.

Каталоги, связанные с ядром

Некоторые каталоги связаны с ядром Linux — самой низкоуровневой частью ОС.

`/boot`
: Файлы для загрузки системы. Здесь находится ядро (обычно в `/boot/vmlinuz` или в файле с похожим именем).

`/lost+found`
: Поврежденные файлы, восстановленные с помощью инструмента восстановления диска.

`/proc`
: Файлы текущих запущенных процессов (для опытных пользователей).

`/sys`
: Файлы для внутренних компонентов ядра (для опытных пользователей).

Файлы в каталогах `/proc` и `/sys` обладают особыми свойствами и относятся к функциям ядра. Файлы в каталоге `/proc` всегда нулевого размера, они доступны только для чтения и датированы сегодняшним днем, однако внутри них находится информация о ядре Linux:

```
→ ls -lG /proc/version
-r--r--r--  1 root  0 Oct  3 22:55 /proc/version
→ cat /proc/version
Linux version 5.15.0-76-generic ...
```

Файлы в каталоге /sys также имеют странный размер и содержат информацию о ядре:

```
→ ls -lG /sys/power/state
-rw-r--r-- 1 root 4096 Jul  8 06:12 /sys/power/state
→ cat /sys/power/state
freeze mem disk
```

В основном файлы в каталогах /proc и /sys используются системными программами, но вы можете их просматривать. Вот несколько примеров.

/proc/ioports	Список аппаратных средств ввода/вывода вашего компьютера
/proc/cpuinfo	Информация о процессоре вашего компьютера
/proc/version	Версия ОС. Команда uname выводит ту же информацию
/proc/uptime	Время работы системы — указывает количество секунд, прошедших с последнего запуска системы. Чтобы получить более удобный результат, используйте команду uptime
/proc/*NNN*	Информация о процессе Linux с идентификатором *NNN*, где *NNN* — это положительное целое число, например /proc/13542
/proc/self	Информация о текущем запускаемом процессе. Символическая ссылка на файл /proc/nnn автоматически обновляется. Попробуйте запустить `→ ls -l /proc/self` Запустите команду несколько раз, и /proc/self каждый раз будет выводить новое значение

Права доступа

В системе Linux может быть зарегистрировано много учетных записей пользователей. Чтобы сохранить конфиденциальность и безопасность, пользователи получают доступ к *некоторым* (не ко всем) файлам системы. Контроль доступа состоит из двух вопросов.

У кого есть права?

У каждого файла и каталога есть *владелец*, который имеет все права. Обычно владельцем файла является пользователь, создавший его. Суперпользователь может сменить владельца файла.

Доступ к файлу может получить определенная *группа* пользователей. Такие группы создаются системным администратором — я рассказываю об этом в разделе «Управление группами» главы 3.

Наконец, файл или каталог может быть доступен для *всех пользователей*, имеющих учетную запись в системе. Набор таких пользователей можно назвать *«другие»* или *«все»*.

Какие есть права?

Создатели файлов, группы пользователей и все прочие могут иметь права на *чтение*, на *запись* (изменение) и на *выполнение* (запуск) определенных файлов. Права могут распространяться на каталоги, которые пользователи могут читать (просматривать входящие в них файлы), записывать (создавать файлы в каталоге и удалять их из него) и выполнять (входить в них с помощью команды `cd`).

Чтобы посмотреть права на файл `myfile`, выполните команду `ls -l`, описанную в разделе «Основные операции с файлами» главы 2:

```
→ ls -l myfile
-rw-r--r-- 1 smith smith  1168 Oct 28  2015 myfile
```

Чтобы посмотреть права на каталог `mydir`, добавьте параметр `-d`:

```
→ ls -ld mydir
drwxr-x--- 3 smith smith  4096 Jan 08 15:02 mydir
```

В приведенном ранее выводе права — это 10 первых символов: буквы `r` (чтение), `w` (запись), `x` (выполнение), другие символы и дефисы. Например:

```
-rwxr-x---
```

Вот что означают эти буквы и символы.

Позиция	Значение
1	**Тип файла:** `-` (дефис) — файл, `d` — каталог (папка), `l` — символическая ссылка, `p` — именованный пайп, `c` — символьное устройство, `b` — блочное устройство

Позиция	Значение
2–4	**Права владельца файла:** r (чтение), w (запись), x (выполнение) или - (отсутствие прав)
5–7	**Права группы пользователей:** r, w, x, -
8–10	**Права остальных пользователей:** r, w, x, -

Мой пример `-rwxr-x---` означает файл, который может читать, записывать и выполнять владелец, читать и выполнять группа пользователей. Другие пользователи не имеют прав на этот файл. Чтобы изменить владельца, группу или права, задействуйте команды `chown`, `chgrp` и `chmod` соответственно. Подробнее об этом написано в разделе «Свойства файлов» главы 2.

Некоторые функции Bash

Оболочка способна на гораздо большее, чем просто выполнение команд. Она *упрощает* выполнение команд с помощью мощных функций: поиска по шаблонам имен файлов, «истории команд» для быстрого вызова предыдущих команд, конвейеров для передачи вывода одной команды на ввод другой, переменных для хранения значений оболочки и т. д. Я рекомендую потратить время на изучение этих возможностей, чтобы повысить эффективность работы. Давайте познакомимся с этими полезными инструментами. (Для получения полной документации запустите функцию `info bash`.)

Какую оболочку вы используете?

В этой книге я предполагаю, что вы работаете с bash. Чтобы узнать, какая оболочка используется, выполните команду

```
→ echo $SHELL
/bin/bash
```

Если ваша оболочка не bash, вы можете запустить команду `bash` напрямую, так как все оболочки — это простые программы (оболочка находится в каталоге `/bin/bash`):

```
→ bash
```

Для возвращения в обычную оболочку выполните команду `exit`. Чтобы сменить стандартную оболочку на bash, прочитайте о команде `chsh` в разделе «Управление учетными записями пользователей» главы 3.

Поиск по шаблону

Шаблон для поиска в оболочке, иногда называемый *подстановочными знаками*, — это сокращение для работы с группами файлов. Например, шаблон `a*` позволяет найти файлы, имена которых начинаются со строчной буквы «a». Оболочка дополнит шаблон до полного набора имен соответствующих файлов. Если вы выполните команду

```
→ ls a*
aardvark  adamantium  apple
```

оболочка расширит шаблон `a*` и добавит имена файлов из текущего каталога, начинающиеся с «a», как если бы вы написали:

```
→ ls aardvark adamantium apple
```

Команда `ls` никогда не узнает, что вы использовали шаблон, — она видит только конечный список имен файлов после расширения. Это означает, что *любая* программа Linux, запускаемая из оболочки, поддерживает шаблоны и другие возможности оболочки. Это очень важный момент. Удивительно, но бóльшая часть пользователей Linux считают, что программы задействуют собственные шаблоны в командной строке, однако это не так. Оболочка делает это *еще до запуска связанной с ней программы*.

В шаблонах нельзя использовать напрямую два специальных символа: начальную точку и слеш каталога (`/`). Эти символы должны указываться как литеральные. Шаблон типа `.bas*` соответствует `.bashrc`, а `/etc/*conf` — всем именам файлов из каталога `/etc`, заканчивающимся на `conf`.

Шаблон	Значение
*	Ноль или более символов, за исключением начальной точки или слеша в имени каталога

Шаблон	Значение
?	Любой одиночный символ, кроме начальной точки или слеша в имени каталога
[*символ*]	Любой одиночный *символ* в указанном диапазоне. Это может быть последовательность символов, например [aeiouAEIOU] для всех гласных, или диапазон с дефисом, например [A-Z] для всех прописных латинских букв, или их комбинация
[!*символ*]	Любой одиночный *символ*, не входящий в указанный диапазон. Например, [!0-9] означает «не цифра»
[^*символ*]	То же, что и [!*символ*]

Файлы с точкой

Имена файлов, начинающиеся с точки, часто скрыты во избежание неразумных и зловредных действий с ними. Примером может служить файл инициализации bash `.bashrc` в вашем домашнем каталоге.

- `ls` не охватывает файлы с точкой в списках каталогов, если вы не используете параметр `-a`.
- При поиске по шаблону начальная точка не учитывается.

В результате файлы с точкой часто называют скрытыми файлами.

Чтобы использовать литеральный дефис в наборе символов, укажите его первым или последним символом так, чтобы он не входил в диапазон. Чтобы включить в диапазон литеральную закрывающую квадратную скобку, укажите ее первой или отделите обратным слешем (`\]`). Чтобы литерально включить символ ^ или !, не указывайте его первым.

Использование скобок

Выражения в фигурных скобках тоже можно применять, и тогда они превратятся в группу аргументов команды. Выражение, разделенное запятыми:

```
{bubble,quick,merge}
```

расширяется сначала до `bubble`, потом до `quick` и, наконец, до `merge` в командной строке:

```
→ echo {bubble,quick,merge}sort.java
bubblesort.java quicksort.java mergesort.java
```

ПРИМЕЧАНИЕ

Ключевое различие между фигурными и квадратными скобками заключается в том, что фигурные скобки применяются с любыми строками, а выражения в квадратных скобках допускаются только с существующими именами файлов.

Фигурные скобки могут расширяться до последовательности значений в диапазоне, если разделить конечные точки диапазона двумя точками (`..`):

```
→ echo {3..12}
3 4 5 6 7 8 9 10 11 12
→ echo {A..E}
A B C D E
→ echo file{1..5}.py
file1.py file2.py file3.py file4.py file5.py
```

Переменные оболочки

Вы можете определить в оболочке переменные и присвоить им значения:

```
→ MYVAR=3          Присвоение переменной MYVAR значения 3
```

Чтобы сослаться на значение переменной, укажите знак доллара перед ее именем:

```
→ echo $MYVAR
3
```

При входе в систему оболочка определяет некоторые стандартные переменные.

Переменная	Значение
`DISPLAY`	Имя дисплея X Window
`HOME`	Ваш домашний каталог, например `/home/smith`

Переменная	Значение
LOGNAME	Ваш логин, например `smith`
MAIL	Ваша входящая электронная почта, например `/var/spool/mail/smith`
OLDPW	Предыдущий каталог вашей оболочки до последней команды `cd`
PATH	Маршрут поиска вашей оболочки — каталоги, разделенные двоеточиями
PWD	Текущий каталог оболочки
SHELL	Путь к вашей оболочке, например `/bin/bash`
TERM	Тип вашего терминала, например `xterm` или `vt100`
USER	Ваше имя для входа в систему

Переменные и их значения по умолчанию ограничены оболочкой, в которой они объявлены. Чтобы переменная и ее значение стали доступны для других программ, вызываемых нашей оболочкой (то есть подпроцессов), используйте команду `export`:

```
→ MYVAR=3
→ export MYVAR
```

или сокращение:

```
→ export MYVAR=3
```

Экспортированная переменная называется *переменной окружения*. Чтобы сделать переменную доступной для каждой новой запускаемой оболочки, заранее добавьте определение переменной в файл конфигурации оболочки. Об этом подробно говорится в разделе «Настройка оболочки» в конце данной главы.

Чтобы отобразить переменные окружения оболочки, выполните команду

```
→ printenv
```

Чтобы переменная окружения была активна только на время выполнения одной команды, добавьте в командную строку *переменная=значение*:

```
→ printenv HOME
/home/smith
→ HOME=/home/sally printenv HOME
/home/sally
→ printenv HOME
/home/smith                    Изначальное значение не изменено
```

Маршрут поиска

Программы могут находиться во всей файловой системе Linux, в основном в каталогах `/bin` и `/usr/bin`. Когда вы выполняете команду, вызывающую программу, то оболочка как-то должна найти эту программу в файловой системе:

```
→ who                          Оболочка должна найти программу who
```

Оболочка ищет программу, обращаясь к переменной окружения `PATH`. Эта переменная представляет собой список каталогов, разделенных двоеточиями. Такой список называется *маршрутом поиска* оболочки:

```
→ echo $PATH
/usr/local/bin:/bin:/usr/bin   Маршрут поиска с тремя каталогами
```

Оболочка ищет файл с именем `who` в каждом из перечисленных каталогов по порядку. Если файл находится в каталоге `who` (скажем, в `/usr/bin/who`), то она запустит программу, а также кэширует местоположение для следующего раза (чтобы узнать больше о кэшировании, выполните команду `hash --help`). В ином случае команда выдаст сообщение об ошибке:

```
bash: who: command not found
```

Чтобы вывести местоположение команды в маршруте поиска, выполните команду `type` или `which`:

```
→ type who
who is /usr/bin/who
→ which who
/usr/bin/who
```

Чтобы временно добавить каталоги в маршрут поиска оболочки, измените переменную `PATH`. Например, добавьте каталог `/usr/sbin` в маршрут поиска оболочки:

```
→ PATH=$PATH:/usr/sbin
→ echo $PATH
/usr/local/bin:/bin:/usr/bin:/usr/sbin
```

Это изменение влияет только на текущую оболочку. Чтобы сохранить его, измените значение `PATH` в файле конфигурации bash — данный процесс описывается в разделе «Настройка оболочки» в конце этой главы. Затем выйдите из системы и снова войдите в нее или запустите файл конфигурации вручную в каждом открытом окне оболочки, например:

```
→ . $HOME/.bashrc          Если вы изменили $HOME/.bashrc
```

Псевдонимы

Команда `alias` определяет удобное сокращение (псевдоним) для другой команды. Например, следующий псевдоним:

```
→ alias ll='ls -lG'
```

определяет новую команду `ll`, выполняющую инструкцию `ls -lG`:

```
→ ll
total 436
-rw-r--r--   1 smith     3584 Oct 11 14:59 file1
-rwxr-xr-x   1 smith       72 Aug  6 23:04 file2
⋮
```

Определите псевдонимы в файле `~/.bashrc` (см. раздел «Настройка оболочки» в конце данной главы), чтобы сделать их доступными для будущих сеансов работы в оболочке[1]. Чтобы вывести список всех псевдонимов, выполните команду `alias`. Если вам нужна бóльшая гибкость, чем псевдонимы, изучите раздел «Программирование с помощью сценариев командной оболочки» главы 6, выполните команду `info bash` и прочитайте раздел «Функции оболочки».

[1] В некоторых случаях для этой цели используется ~/.bash_aliases.

Встроенные команды

Большинство команд Linux — это программы в файловой системе Linux. Примерами служат команды `wc` и `who`, обычно находящиеся в каталоге `/usr/bin`. Оболочка находит и запускает их с помощью переменной `PATH`. Я уже рассказывал об этом в разделе «Маршрут поиска» ранее в этой главе. Однако некоторые команды являются встроенными функциями оболочки и называются *встроенными командами*. В этой главе вы уже познакомились с несколькими встроенными командами, например `cd`, `alias` и `export`. Чтобы определить тип команды, выполните команду `type`:

```
→ type wc cd ll                    Вывод типов перечисленных команд
wc is /usr/bin/wc                  Программа в файловой системе
cd is a shell builtin              Встроенная команда оболочки
ll is aliased to `ls -lG'          Псевдоним
```

Ввод, вывод и перенаправление

Большинство команд Linux принимают данные с ввода и/или передают данные на вывод. Ввод с клавиатуры является *стандартным вводом*, или *stdin*. Вывод на экран называется *стандартным выводом*, или *stdout*. Сообщения об ошибках обрабатываются особым образом и выводятся в стандартный поток вывода сообщений об ошибке, или *stderr*, который обычно представлен экраном. Однако в Linux стандартный вывод отделен от вывода сообщений об ошибке[1].

Оболочка может перенаправлять стандартный ввод, стандартный вывод и стандартный вывод сообщений об ошибке в файлы и прочесть данные из них. Другими словами, любая команда, считывающая данные со стандартного ввода, может получить входные данные из файла с помощью оператора <:

```
→ команда < файл_источник
```

[1] Так, вы можете захватить стандартный вывод в файл, но при этом на экране будут отображаться сообщения о стандартных ошибках.

Аналогично, любая команда, которая пишет результат в стандартный вывод, может выводить данные в файл:

```
→ команда > файл_назначения      Создание/перезапись файла вывода
→ команда >> файл_назначения     Добавление в файл вывода
```

Команда, которая выводит сообщение об ошибке, может перенаправить вывод в файл, оставив стандартный вывод нетронутым:

```
→ команда 2> журнал_ошибок
```

Чтобы перенаправить стандартный вывод и стандартные ошибки в соответствующие файлы, выполните следующее:

```
→ команда > файл_назначения 2> журнал_ошибок   Отдельные файлы
→ команда &> файл_назначения                   Один файл (предпочтительно)
→ команда >& файл_назначения                   Один файл (используется реже)
```

Комбинированные команды

Bash позволяет выйти за рамки простых команд и объединить несколько программ в одной командной строке.

Последовательность команд

Чтобы вызвать несколько команд последовательно в одной командной строке, разделите их точкой с запятой:

```
→ команда1 ; команда2 ; команда3
```

Чтобы запустить последовательность команд как обычно, но остановить их выполнение в случае, если какая-то из команд не выполняется, разделите их символами && (И):

```
→ команда1 && команда2 && команда3
```

Чтобы запустить последовательность команд и остановить их выполнение при достижении успеха хотя бы одной из них, разделите их символами || (ИЛИ):

```
→ команда1 || команда2 || команда3
```

Конвейеры

Вы можете перенаправить стандартный вывод одной команды на стандартный ввод другой, используя оператор вертикальной черты (|) оболочки — пайп. (Чтобы напечатать этот символ, нажмите сочетание клавиш Shift+\ в английской раскладке.)

Например, команда

```
→ who | sort
```

передает вывод who команде sort, выводя отсортированный по алфавиту список вошедших в систему пользователей. Вы также можете применить несколько пайпов. Давайте снова отсортируем вывод who, извлечем первый столбец информации (с помощью awk) и выведем все результаты постранично (с помощью команды less):

```
→ who | sort | awk '{print $1}' | less
```

Подстановка команд

Если вы заключите команду в обратные кавычки, оболочка выполнит команду и заменит ее выводом:

```
→ date +%Y                                  Печать текущего года
2024
→ echo This year is `date +%Y`
This year is 2024
```

Знак доллара и круглые скобки эквивалентны обратным кавычкам:

```
→ echo This year is $(date +%Y)
This year is 2024
```

и они могут быть вложены друг в друга:

```
→ echo Next year is $(expr $(date +%Y) + 1)
Next year is 2025
```

Подстановка процессов

Некоторые программы не очень хорошо работают в конвейерах, так как считывают данные не со стандартного ввода, а только из файлов на диске. Примером может служить

команда `diff`, сравнивающая два файла построчно и выводящая их различия. *Подстановка процесса* — это способ заставить команду типа `diff` считывать данные со стандартного ввода. Этот процесс запускает программу и позволяет ее выводу маскироваться под файл, с которым работают программы типа `diff`. С помощью оператора подстановки процессов `<()` можно сравнивать вывод двух команд, а не двух файлов на диске.

Предположим, у вас есть каталог с парами файлов JPG и TXT:

```
→ ls jpegexample
file1.jpg  file2.jpg  file3.jpg  ...
file1.txt  file2.txt  file3.txt  ...
```

и вы хотите убедиться, что каждому графическому файлу соответствует текстовый файл, и наоборот. Можно создать два временных файла, один из которых содержит имена графических файлов, а другой — текстовых, удалить расширения файлов с помощью команды `cut` и сравнить два временных файла с помощью команды `diff`:

```
→ cd jpegexample
→ ls *.jpg | cut -d. -f1 > /tmp/jpegs
→ ls *.txt | cut -d. -f1 > /tmp/texts
→ diff /tmp/jpegs /tmp/texts
5a6
> file6                                  Нет файла file6.jpg
8d8
< file9                                  Нет файла file9.txt
```

Подстановка процессов выполняет ту же задачу с помощью одной команды и без временных файлов:

```
→ diff <(ls *.jpg|cut -d. -f1) <(ls *.txt|cut -d. -f1)
```

Каждый оператор `<()` означает имя файла в командной строке, как если бы этот «файл» содержал вывод `ls` и `cut`.

Особенности интерпретации

Оболочка интерпретирует каждый символ команды. Чтобы предотвратить интерпретацию, заключайте символы в кавычки или экранируйте.

Заключение в кавычки

Обычно оболочка воспринимает пробельные символы как разделитель строк в командной строке. Чтобы правильно интерпретировать строку, *содержащую* пробельный символ (например, имя файла с пробелом), поставьте с обоих концов одинарные или двойные кавычки. Тогда оболочка воспримет строку как единое целое. Благодаря одинарным кавычкам содержимое строки воспринимается литерально, а двойные кавычки позволяют интерпретировать переменные и другие конструкции оболочки:

```
→ echo 'The variable HOME has value $HOME'
The variable HOME has value $HOME
→ echo "The variable HOME has value $HOME"
The variable HOME has value /home/smith
```

Экранирование

Если символ интерпретируется оболочкой, но вы хотите использовать его литерально (например, если * — это литеральная звездочка, а не шаблон имени файла), то поставьте перед ним обратный слеш (\). Это называется *экранированием* специального символа:

```
→ echo a*                              Шаблон файла
aardvark adamantium apple
→ echo a\*                             Литеральная звездочка
a*
→ echo "I live in $HOME"               Вывод значения переменной
I live in /home/smith
→ echo "I live in \$HOME"              Вывод литерального знака доллара
I live in $HOME
```

Вы также можете экранировать управляющие символы (знаки табуляции, перевода строк, ^D и т. д.), вводя перед ними ^V. Это особенно полезно для символов табуляции, которые в противном случае оболочка использовала бы для завершения имени файла (см. раздел «Автозавершение имен файлов» далее в этой главе):

```
→ echo "There is a tab between here^V      and here"
There is a tab between here      and here
```

Редактирование в командной строке

В bash можно выполнять операции редактирования в командной строке с помощью клавиатуры по аналогии с текстовыми редакторами Emacs и Vim (см. раздел «Создание и редактирование файлов» главы 2). Чтобы редактирование в командной строке с помощью клавиш Emacs было возможно, выполните следующую команду (напишите ее в файле конфигурации bash, чтобы она работала постоянно):

```
→ set -o emacs
```

Команда для поддержки управления в духе vi (Vim):

```
→ set -o vi
```

Сочетание Emacs	Сочетание Vim (после Esc)	Описание
^P или ↑	k или ↑	Переход к предыдущей команде
^N или ↓	j или ↓	Переход к следующей команде
^R		Интерактивный поиск предыдущей команды
^F или →	l или →	Переход на один символ вперед
^B или ←	h или ←	Переход на один символ назад
^A	0	Переход к началу строки
^E	$	Переход к концу строки
^D	x	Удаление следующего символа
^U	^U	Удаление до начала строки

История команд

Оболочка хранит предыдущие команды и позволяет повторно выполнить их. Эта функция называется *историей команд*. Попробуйте воспользоваться следующими полезными командами и выражениями.

Команда	Значение
`history`	Вывод истории команд
`history` *N*	Вывод последних *N* команд в истории

Команда	Значение
`history -c`	Очистка (удаление) истории
`!!`	Вывод предыдущей команды. Чтобы выполнить команду повторно: `→ !! <Enter>`
`!N`	Вывод *N* команд в истории
`!-N`	Вывод команды, выполненной *N* команд назад

`!$`	Вывод последнего аргумента предыдущей команды. Команда подходит для проверки наличия файлов перед выполнением деструктивной команды, например `rm`:
	`→ ls z*`
	`zebra.txt  zipfile.zip  zookeeper`
	`→ rm !$` *То же самое, что и rm z**

`!*`	Вывод всех аргументов предыдущей команды:
	`→ ls myfile emptyfile hugefile`
	`emptyfile  hugefile  myfile`
	`→ wc !*`
	`  18  211  1168 myfile`
	`  0  0  0 emptyfile`
	`333563  2737540  18577839 hugefile`
	`333581  2737751  18579007 total`

Автозавершение имен файлов

Во время ввода имени файла нажмите клавишу Tab, и тогда оболочка допишет имя автоматически. Если введенное вами имя совпадает с именами нескольких файлов, то оболочка подаст звуковой сигнал, указывающий на совпадение. Нажмите клавишу Tab второй раз, и тогда оболочка покажет альтернативные варианты имен. Попробуйте:

```
→ cd /usr/bin
→ ls un<Tab><Tab>
```

Оболочка отобразит все имеющиеся в папке `/usr/bin` файлы, начинающиеся с `un`, например `uniq` и `unzip`. Введите еще несколько символов, чтобы сузить выбор, и снова нажмите клавишу Tab.

Управление заданиями оболочки

`jobs`	Вывод списка заданий
`&`	Указывается после команды и запускает ее в фоновом режиме
`^Z`	Сочетание клавиш для остановки выполнения текущего задания
`suspend`	Перевод задания в оболочке в состояние ожидания
`fg`	Отмена перевода задания в состояние ожидания
`bg`	Продолжение выполнения задания в основном режиме
`disown`	Отмена задания

Все оболочки Linux имеют функцию *управления заданиями*. Она позволяет выполнять команды в фоновом (для многозадачности) и основном режимах. *Задание* — это единица работы оболочки. Когда вы выполняете команду в интерактивном режиме, текущая оболочка считает ее заданием. Когда команда завершается, связанное с ней задание исчезает. Задания находятся на более высоком уровне, чем процессы Linux: ОС Linux даже не подозревает о существовании процессов, так как они являются конструкциями оболочки. Вот несколько важных терминов, относящихся к управлению заданиями.

Основное задание

Это выполняемое задание, которое занимает приглашение оболочки, и вы не можете выполнить другую команду.

Фоновое задание

Это выполняемое задание, которое не занимает приглашение оболочки, поэтому можно выполнять другие команды в этой же оболочке.

Перевод в состояние ожидания

Временная остановка выполнения (подвешивание) основного задания.

Возобновление

Возобновление работы ожидающего задания. Оно станет приоритетным и будет выполняться в качестве основного.

Прекращение выполнения

Сообщение оболочке о прекращении отслеживания задания. Основные процессы продолжают выполняться.

jobs

stdin **stdout** -file --opt **--help** --version

Встроенная команда `jobs` отображает по номеру и имени задания, выполняемые в текущей оболочке:

```
→ jobs
[1]-  Running         emacs myfile &      Фоновое задание
[2]+  Stopped         ssh example.com     Приостановленное задание
```

Целое число слева — это номер задания, а знак «плюс» обозначает задание по умолчанию, на которое влияют команды `fg` (основное) и `bg` (фоновое).

&

Амперсанд, указанный в конце команды, переводит ее выполнение в фоновый режим:

```
→ emacs myfile &
[2] 28090
```

Ответ оболочки содержит номер задания (`2`) и идентификатор процесса команды (`28090`).

^Z

Нажатие сочетания клавиш ^Z во время выполнения активной задачи прекращает выполнение. Процесс выполнения задания просто останавливается, но его состояние запоминается:

```
→ sleep 10                              Ожидает 10 секунд
^Z
[1]+  Stopped    sleep 10
→
```

Теперь вы можете выполнить команду `bg`, чтобы перевести команду `sleep` в фоновой режим, или команду `fg`, чтобы возобновить выполнение команды в качестве основной. Также можете оставить ее в этом состоянии и выполнять другие команды.

suspend

stdin stdout -file --opt **--help** --version

Встроенная команда `suspend` приостанавливает работу текущей оболочки, если это возможно. Например, если с помощью `sudo`-команды вы создали оболочку суперпользователя и хотите вернуться в первоначальную оболочку, то команда `suspend` приостановит оболочку суперпользователя:

```
→ whoami
smith
→ sudo bash                    Запуск оболочки суперпользователя
[sudo] password: xxxxxxxx
# whoami
root
# suspend                      Остановка работы оболочки суперпользователя
[1]+  Stopped    sudo bash
→ whoami                       Возвращение к первоначальной оболочке
smith
```

bg

stdin stdout -file --opt **--help** --version

```
bg [%задание]
```

Встроенная команда `bg` возобновляет выполнение приостановленного *задания* в фоновом режиме. Без аргументов команда `bg` обращается к последнему приостановленному заданию. Чтобы указать конкретное задание (показанное командой `jobs`), введите номер или имя задания, поставив перед ним знак процента:

```
→ bg %2                        Выполнение задания 2 в фоновом режиме
→ bg %cat                      Передача задания с именем, начинающимся с cat
```

Некоторые задания не могут выполняться в фоновом режиме, например, если они ожидают ввода. Если вы попытаетесь оставить их в фоновом режиме, то оболочка приостановит задание и выведет следующее сообщение:

```
[2]+  Stopped        Команда
```

Теперь возобновим задание (с помощью команды `fg`).

---

fg

stdin stdout -file --opt **--help** --version

```
fg [%задание]
```

Встроенная команда `fg` возобновляет приостановленное или находящееся в фоновом режиме *задание*. Если не указаны аргументы, то команда обращается к последнему остановленному или фоновому заданию. Чтобы указать конкретное задание (как показано в команде `jobs`), введите его номер или имя, поставив перед ним знак процента:

```
→ fg %2          Возобновление задания 2 в качестве основного
→ fg %cat        Возобновление в качестве основного задания с именем,
                 начинающимся с cat
```

---

disown

stdin stdout -file --opt **--help** --version

```
disown [-ar] [-h] [%задание]
```

Встроенная команда `disown` отменяет указанное *задание* в текущей оболочке. Процессы Linux, относящиеся к заданию, продолжат выполняться — вы просто больше не можете управлять им с помощью команд `bg`, `fg`, `jobs` и т. д. Это полезно для длительных заданий, с которыми вам не нужно постоянно взаимодействовать, или таких, которые должны продолжать выполняться после выхода из оболочки (см. команду `nohup` в разделе «Управление процессами» главы 3»):

```
→ disown %2          Отмена задания 2
→ disown %cat        Отмена задания с именем, начинающимся с cat
→ disown -h %2       Продолжение выполнения задания 2 после выхода из оболочки
→ disown -r          Отмена всех выполняющихся заданий
→ disown -a          Отмена всех заданий
```

Параллельный запуск нескольких оболочек

Управление заданиями помогает контролировать несколько команд одновременно, но только одна из них может выполняться в качестве основной. Более того, вы можете одновременно запускать несколько оболочек, в каждой из которых будет выполняться одна основная команда и несколько фоновых.

Если на вашем компьютере Linux установлена графическая система, например KDE или GNOME, вы можете запускать множество оболочек одновременно, открывая несколько окон оболочки (см. раздел «Запуск оболочки» ранее в этой главе). К тому же некоторые графические программы оболочки, например `konsole` в KDE, позволяют открывать в одном окне несколько вкладок, на каждой из которых будет запущена собственная копия оболочки.

Даже без графической системы — например, через сетевое соединение SSH — вы все равно можете управлять несколькими оболочками одновременно. Команда `tmux` имитирует несколько окон оболочки в обычном ASCII-терминале. С помощью специальных клавиш можно переключаться с одного виртуального окна на другое. (Подобной программой является `screen`, но я рекомендую команду `tmux` — она лучше поддерживается и удобнее.) Чтобы начать работу с `tmux`, выполните команду

```
→ tmux
```

Запустится новая оболочка с дополнительной строкой состояния в нижней части терминала. Это означает, что вы работаете с одним виртуальным окном. По умолчанию программа `tmux` предоставляет десять таких окон, обозначенных числами от 0 до 9. В ходе работы можно переключаться между ними. В каждом окне запускается одна оболочка, но вы можете разделить окно на несколько вкладок, чтобы запускать сразу несколько оболочек. Чтобы работать с `tmux`, попробуйте использовать следующие сочетания клавиш.

1. В текущем окне `tmux` запустите команду `ls`.
2. Нажмите сочетание клавиш ^Bc (сначала нажмите Ctrl+B, затем — клавишу C). `tmux` отобразит новое приглашение оболочки во втором виртуальном окне. В строке состояния будут указаны два виртуальных окна под номерами 0 и 1.
3. Во втором окне выполните любую другую команду, например `df`.
4. Нажмите сочетание клавиш ^Bn, чтобы переключиться на окно 0, где снова отображается вывод команды `ls`.
5. Нажмите сочетание клавиш ^Bn еще несколько раз, чтобы переключаться между двумя виртуальными окнами.
6. Нажмите сочетание клавиш ^B%, чтобы разделить текущее окно на две вкладки.
7. Нажмите сочетание клавиш ^B», чтобы разделить текущую вкладку на две по вертикали. Теперь у вас будет три оболочки на отдельных вкладках.

Большинство аспектов работы `tmux` настраивается в файле `~/.tmux_conf`. Там вы даже можете выбрать для сочетания клавиш ^B любую другую буквенную клавишу. В таблице далее я привожу стандартные значения сочетаний клавиш.

Сочетание клавиш	Значение
^B?	Запуск интерактивной справки. Для выхода нажмите клавишу Q
^Bc	Создание нового окна
^B0, ^B1… ^B9	Открыть окно определенного номера от 0 до 9
^Bn	Переход к следующему окну по порядку
^Bp	Переход к предыдущему окну по порядку
^Bl	Переход к недавно использованному окну
^B%	Разделение окна по вертикали
^B"	Разделение окна по горизонтали
^Bo	Переход к следующей вкладке
^B ←	Переход к вкладке слева

Сочетание клавиш	Значение
^B →	Переход к вкладке справа
^B ↑	Переход к вкладке сверху
^B ↓	Переход к вкладке снизу
^Bq	Отображение номеров вкладок
^Bx	Закрытие текущей вкладки
^B^B	Применение исходного предназначения сочетания клавиш Ctrl+B, игнорируя настроенное в `tmux`
^B^Z	Остановка работы `tmux`
^Bd	Отключение от сессии `tmux` и возвращение в исходную оболочку. Для возврата в `tmux` выполняется команда `tmux attach`
^D	Завершение работы оболочки в окне или на вкладке. Это обычное завершение работы файла (описано в разделе «Завершение работы оболочки» далее в этой главе), закрывающее любую оболочку
^B:kill-session	Закрытие всех окон и прекращение работы `tmux`

Несколько замечаний о работе `tmux`.

- Если в оболочке `tmux` не поддерживаются заданные вами псевдонимы, переменные или другие настройки оболочки, то дело в том, что `tmux` запускает оболочку без учета файла инициализации `.bashrc`. `tmux` использует только файл запуска (`.bash_profile`, `.bash_login` или `.profile`). Чтобы решить эту проблему, добавьте в файл запуска следующие строки:

```
# Код моего файла .bashrc
if [ -f ~/.bashrc ]; then
  . ~/.bashrc
fi
```

- Если вы работаете в текстовом редакторе, `tmux` перехватывает все нажатия Ctrl+B, даже предназначенные для команд редактирования. Нажмите ^B^B, чтобы передать редактору предустановленное значение сочетания клавиш Ctrl+B.

- Не запускайте `tmux` локально на графическом Рабочем столе, вместо этого запустите несколько окон из оболочки. Это намного проще, и вы избежите следующей проблемы: если настроили оболочку на выполнение команд при выходе из системы (например, в файле `~/.bash_logout`), то оболочки `tmux` будут выполнять эти команды при выходе, даже если вы не уходили с Рабочего стола. Такая ситуация может привести к нежелательным последствиям.

Прерывание выполняемых команд

Чтобы немедленно прервать выполняемую команду, нажмите сочетание клавиш ^C. В примере далее я прерываю команду `cat`, выводящую объемный файл:

```
→ cat hugefile
Lorem ipsum dolor sit amet, consectetur adipiscing odio.
Praesent libero.
Sed cursus ante dapibus diam. quis sem at nibh elementum
blah blah blah ^C
→
```

Чтобы прервать фоновую программу, сделайте ее основной с помощью команды `fg`, а затем нажмите сочетание клавиш ^C:

```
→ sleep 50 &
[1] 12752
→ jobs
[1]-  Running       sleep 50 &
→ fg %1
sleep 50
^C
→
```

или выполните команду `kill`, описанную в разделе «Управление процессами» главы 3. Сочетание клавиш ^C — это функция оболочки. Оно не влияет на «захваченные» сочетанием клавиш ^C программы и не завершает их работу. Для прерывания работы текстовых редакторов и приложений с графическим интерфейсом используйте команду `kill`.

Восстановление оболочки после прерывания команды

Прерывание команды с помощью сочетания клавиш ^C может повредить вашу оболочку, так как прерванная программа может закрыться некорректно. Распространенный симптом — неработоспособность настроенных сочетаний клавиш. Чтобы восстановить оболочку, выполните следующие действия.

1. Нажмите сочетание клавиш ^J, чтобы отобразилось приглашение оболочки. Это сочетание клавиш сработает, даже если нет отклика на нажатие клавиши Enter.
2. Введите команду `reset` (даже если во время ввода не отображаются буквы) и нажмите сочетание клавиш ^J, чтобы выполнить команду. Ваша оболочка должна вернуться в нормальное состояние.

Завершение работы оболочки

Чтобы завершить работу оболочки, выполните команду `exit`:

```
→ exit
```

или нажмите сочетание клавиш ^D. Оно посылает сигнал «конец файла» любой программе, считывающей стандартный ввод. Это касается и самой оболочки.

Настройка оболочки

Поведением оболочки bash управляют несколько файлов в вашем домашнем каталоге. Файлы инициализации `.bash_profile`, `.bash_login` и `.pro-file` содержат команды, которые выполняются при каждом запуске системы. (Выберите один файл запуска и используйте только его. Я рекомендую файл `.bash_profile`, так как некоторые другие оболочки также используют `.profile`.) Команды в файле инициализации `.bashrc` выполняются каждый раз, когда вы запускаете интерактивную оболочку. А команды в файле

`.bash_logout` запускаются каждый раз, когда вы выходите из системы. Все эти файлы могут определять переменные, запускать программы, печатать бессмысленные сообщения и т. д. Другие оболочки Linux используют иные конфигурационные файлы, как показано в табл. 1.1.

Таблица 1.1. Файлы конфигурации оболочки в $HOME и время их чтения

Оболочка	При входе	С помощью других интерактивных оболочек	При выходе
bash	.bash_profile, .bash_login, .profile	.bashrc	.bash_logout
dash	.profile		
fish*	.config/fish /config.fish	.config/fish /config.fish	
ksh	.profile, .kshrc	.kshrc	
tcsh	.tcshrc, .cshrc, .login	.tcshrc, .cshrc	
zsh*	zshenv, .zprofile, .zlogin	.zshenv, .zshrc	.zlogout

* Чтобы изменить пути к файлам с помощью переменных окружения, см. документацию.

Другие файлы конфигурации оболочки находятся в каталоге `/etc` для управления всей системой (см. соответствующую *страницу man-документации* для используемой оболочки). Все эти файлы конфигурации являются примерами *сценариев командной оболочки* — это исполняемые файлы, содержащие команды оболочки. Более подробно об этом рассказывается в разделе «Программирование с помощью сценариев командной оболочки» главы 6.

Помощь

Если вы не можете найти искомую информацию в этой книге, вот несколько способов получения помощи.

Выполните команду `man`

Команда `man` выводит документацию по указанной программе. Например, чтобы узнать, как подсчитать слова в файле с помощью программы `wc`, выполните следующую команду `man`:

```
→ man wc
```

Чтобы найти документацию (man-страницы) по определенной теме с помощью ключевого слова, укажите параметр `-k` вместе с ключевым словом:

```
→ man -k database
```

Если информация не помещается на одном экране, можете выводить результат постранично с помощью команды `less` (нажмите клавишу Q, чтобы завершить выполнение команды):

```
→ man -k database | less
```

Выполните команду `info`

Команда `info` — это расширенная гипертекстовая справочная система, охватывающая многие команды Linux:

```
→ info ls
```

Во время выполнения команды `info` можете нажать несколько полезных клавиш:

- для получения помощи — H;
- для выхода — Q;
- для перехода вперед или назад — Пробел и Backspace соответственно;
- для перехода между гиперссылками — Tab;
- для перехода по гиперссылке — Enter.

Если в `info` нет документации по запрашиваемой команде, то она отобразит man-страницу команды. Чтобы получить список доступной документации, нужно ввести `info`. Чтобы получить дополнительную информацию по команде `info`, введите `info info`.

Используйте параметр `--help` *(если он поддерживается)*

Многие команды Linux поддерживают параметр `--help` или `-h` и в качестве результата выдают короткое справочное сообщение. Попробуйте:

```
→ wc --help
```

Если вывод длиннее одного экрана, то можно разбить его на страницы с помощью команды `less`:

```
→ wc --help | less
```

Изучите каталог `/usr/share/doc`

Этот каталог содержит вспомогательные документы для программ, все они организованы по имени и версии. Например, файлы для редактора Emacs версии 28 можно найти (в зависимости от дистрибутива) в каталоге `/usr/share/doc/emacs28`.

Веб-сайты, посвященные конкретным дистрибутивам

У большинства дистрибутивов есть официальный сайт с документацией, дискуссионными форумами и другими ресурсами. Чтобы найти его, введите в браузере название конкретного дистрибутива, например Ubuntu. Вики Arch Linux (https://oreil.ly/98iAg) особенно информативна вне зависимости от вашего дистрибутива.

Справочные сайты

Можете задать вопросы об операционной системе Linux на сайтах cunix.stakexchange.org, linuxquestions.org, itsfoss.community и nixcraft.com.

Поиск в интернете

Если вы не понимаете сообщение об ошибке Linux, скопируйте номер ошибки и найдите ее в интернете. Можете заключить текст ошибки в двойные кавычки для уточнения поиска.

На этом я завершаю обзор Linux и оболочки. В следующих главах мы будем рассматривать конкретные команды Linux для работы с файлами, процессами, пользователями, сетью, мультимедиа и т. д.

ГЛАВА 2

Команды для работы с файлами

Основные операции с файлами

`ls`	Вывод содержимого каталога
`cp`	Копирование файла
`mv`	Перемещение (переименование) файла
`rm`	Удаление файла
`ln`	Создание ссылки на файл (альтернативных имен)

Манипулирование файлами — одно из первых действий в системе Linux. Вам придется копировать, переименовывать, удалять файлы и т. д.

ls

stdin **stdout** -file **--opt** **--help** **--version**

`ls [`*`параметр(ы)`*`] [`*`файл(ы)`*`]`

Команда `ls` выводит список атрибутов файлов и каталогов. Вы можете вывести файлы, находящиеся в текущем каталоге:

```
→ ls
```

в указанных каталогах:

```
→ ls каталог1 каталог2 каталог3
```

или найти определенные файлы:

```
→ ls файл1 файл2 файл3
```

СОВЕТ

Если команда ls работает некорректно, возможно, в вашем дистрибутиве для нее определен псевдоним (см. раздел «Псевдонимы» главы 1). Чтобы проверить наличие псевдонима, выполните следующую команду:

```
→ alias ls
Alias ls='/bin/ls –FHN'                    Да, есть псевдоним
```

Чтобы запустить оригинальную команду, а не псевдоним, укажите перед ее именем обратный слеш (\ls). Чтобы удалить псевдоним, выполните команду unalias ls. Затем в файле конфигурации оболочки найдите определение псевдонима (см. раздел «Настройка оболочки» главы 1) и удалите его. Если вы не можете найти определение псевдонима, то либо добавьте команду unalias ls, либо определите новый псевдоним, который будет работать так, как вам нужно, например alias ls="/bin/ls".

Наиболее важными параметрами `ls` являются `-a`, `-l` и `-d`. По умолчанию `ls` не показывает файлы, имена которых начинаются с точки (подробнее об этом говорится во врезке «Файлы с точкой» в главе 1). Параметр `-a` отображает все файлы. В зависимости от настроек учетной записи `ls` будет выводить все файлы с точкой в начале списка (сортировка по точке), а все прочие файлы — в алфавитном порядке после точки:

```
→ ls
myfile   myfile2
→ ls -a
.hidden_file   myfile   myfile2
```

Параметр `-l` выдает длинное описание:

```
→ ls -l myfile
-rw-r--r--  1 smith users  1168 Oct 28 2015 myfile
```

Слева направо: права доступа к файлу (`-rw-r-- r--`), количество жестких ссылок (`1`), владелец (`smith`), группа (`users`), размер (1168 байт), дата последнего изменения (28 октября 2015 года) и имя файла. Дополнительную информацию об атрибутах см. в разделе «Права доступа» главы 1.

Параметр `-d` выводит информацию о самом каталоге:

```
→ ls -ld dir1
drwxr-xr-x  1 smith users  4096 Oct 29 2015 dir1
```

Полезные параметры

`-a`	Вывод списка всех файлов, включая те, имена которых начинаются с точки
`-l`	Длинное описание, включая атрибуты. Чтобы вывести размеры файлов в килобайтах, мегабайтах и гигабайтах, а не в байтах, используется параметр `-h`
`-h`	В длинном описании вместо байтов будут выводиться килобайты, мегабайты и другие понятные пользователю обозначения
`-G`	В длинном описании не будет выводиться информация о принадлежности файла к группам
`-F`	Добавление к именам файлов значимых символов, указывающих на их тип. Добавление / к каталогам, * — к исполняемым файлам, @ — к символическим ссылкам, | — к именованным каталогам и = — к сокетам. Это лишь визуальные индикаторы, а не часть имен файлов!
`-S`	Сортировка файлов по размеру
`-t`	Сортировка файлов по времени последнего изменения
`-r`	Обращение порядка сортировки
`-R`	Отображение списка из подкаталогов путем рекурсивного вывода
`-d`	Вывод сведений о каталоге без вывода его содержимого

---

cp

stdin stdout -file **--opt** **--help** **--version**

```
cp [параметр] источник назначение
cp [параметр] (файл | каталог) каталог
```

Команда `cp` копирует один *файл* в *другой_файл*:

```
→ cp файл другой_файл
```

или копирует несколько файлов в указанный *каталог*:

```
→ cp файл1 файл2 файл3 каталог
```

Используйте параметр `-a` или `-r` для рекурсивного копирования, включая все подкаталоги и их содержимое. Для более тонкого копирования см. команду `rsync` в разделе «Резервное хранение и удаленное хранение» главы 4.

Полезные параметры

`-p` Копирование содержимого файла и прав доступа, меток времени и, если у вас достаточно прав, владельца и группы. (В противном случае владельцем копии станете вы, метка времени будет заменена текущей датой, а права доступа установлены в соответствии с настройками пользовательской маски в вашей оболочке.)

`-a` Копирование иерархии каталогов в обратном порядке с сохранением всех атрибутов файлов и ссылок

`-r` Копирование иерархии каталогов в обратном порядке. Параметр не сохраняет атрибуты файлов, а также права доступа и метки времени. Он сохраняет символические ссылки

`-i` Интерактивный режим. Создает запрос перед перезаписью файлов назначения

`-f` Принудительное копирование. Если целевой файл существует, он будет перезаписан

---

mv

stdin stdout -file **--opt** **--help** **--version**

`mv [`*`параметр`*`] `*`источник назначение`*

Команда `mv` переименовывает файл:

```
→ mv файл-источник файл-назначение
```

или перемещает файлы и каталоги в каталог назначения:

```
→ mv файл1 файл2 каталог1 каталог2 каталог_назначения
```

Полезные параметры

`-i` Интерактивный режим. Создает запрос перед перезаписью целевых файлов

`-f` Принудительное перемещение. Если целевой файл существует, он будет перезаписан

---

rm

stdin stdout -file **--opt** **--help** **--version**

`rm [`*`параметр`*`] `*`файл`*` | `*`каталог`*

Команда `rm` удаляет файлы:

```
→ rm файл1 файл2
```

или удаляет каталоги в обратном порядке:

```
→ rm -r каталог1 каталог2
```

ВНИМАНИЕ!

Используйте команду rm -r с осторожностью: она может быстро уничтожить большое количество файлов, особенно в сочетании с параметром -f (игнорирует ошибки и не отображает подсказки).

Применяйте команду sudo rm -r с особой осторожностью: она может сделать операционную систему неработоспособной.

Полезные параметры

`-f`	Интерактивный режим. Запрашивает подтверждение, прежде чем начать управлять каждым файлом
`-i`	Принудительное удаление, игнорирует ошибки и предупреждения
`-r`	Рекурсивное удаление каталога и его содержимого

ln

stdin stdout -file --opt --help --version

`ln [`*`параметр`*`] `*`источник назначение`*

Команда `ln` создает *ссылку*, позволяющую файлу располагаться в нескольких местах файловой системы одновременно. Существует два вида ссылок (рис. 2.1). *Жесткая ссылка* — это второе имя для одного и того же физического файла на диске. На техническом жаргоне обе ссылки относятся к одному и тому же *индексному дескриптору* — структуре данных, определяющей местоположение содержимого файла на диске. Показанная далее команда создает жесткую ссылку `myhardlink` на файл `myfile`:

```
→ ln myfile myhardlink
```

Символическая ссылка (иногда называется гибкой ссылкой) — это указатель на *путь* (не на индексный дескриптор) другого файла или каталога. Если вы знакомы с ярлыками Windows или псевдонимами macOS, то у вас не будет проблем с символическими ссылками — они очень похожи. Чтобы создать символическую ссылку, добавьте параметр `-s`:

```
→ ln -s myfile mysoftlink
```

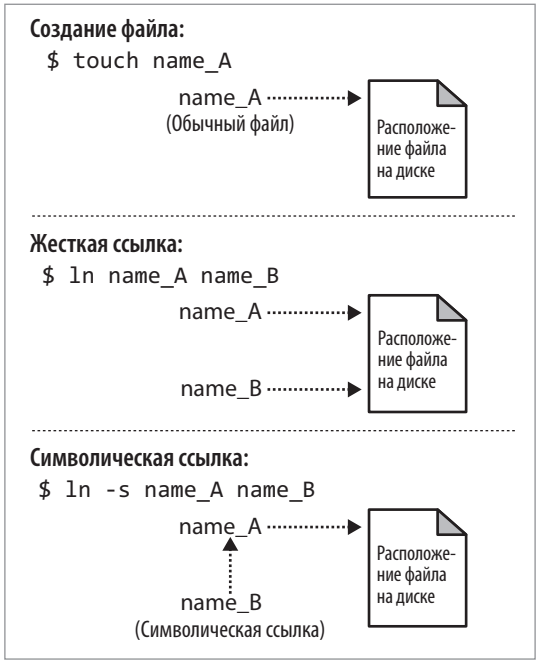


Рис. 2.1. Жесткая ссылка указывает на данные файла. Символическая ссылка указывает на путь к файлу

Жесткие и символические ссылки имеют важные различия.

- Если вы переименуете или удалите исходный файл, жесткая ссылка не изменится — она по-прежнему будет указывать на те же *данные*, что и исходный файл. А вот символическая ссылка перестанет работать — она будет указывать на *путь* к исходному файлу, которого больше не существует. Если вы используете некорректную символическую ссылку в команде, то получите ошибку «Файл не найден».
- Жесткие ссылки могут существовать только на том же устройстве, что и исходный файл. Символические ссылки могут указывать на файлы на других устройствах, так как указывают пути к файлам, а не их данные.

- Символические ссылки могут указывать на каталоги, а жесткие ссылки, как правило, нет. (В некоторых файловых системах суперпользователь может создавать жесткую ссылку на каталог с помощью параметра `-d`.)

Полезные параметры

`-s`	Создание символической ссылки вместо жесткой
`-i`	Интерактивный режим. Перед записью целевых файлов запрашивается подтверждение
`-f`	Принудительное создание ссылки. Если файл уже существует, то он будет перезаписан
`-b`	Создание резервной копии. Если файл уже существует, он переименовывается с добавлением символа «тильда», а затем создается ссылка
`-d`	Создание жесткой ссылки на каталог, если это возможно (только для суперпользователей)

Чтобы узнать, куда указывает символическая ссылка, выполните одну из следующих команд, показывающих, что ссылка `examplelink` указывает на файл `myfile`:

```
→ readlink examplelink
myfile
→ ls -l examplelink
lrwxrwxrwx 1 smith     ...     examplelink -> myfile
```

Символические ссылки могут указывать на другие символические ссылки. Чтобы проследить всю цепочку ссылок и понять, куда они ведут, используйте команду `readlink -f`.

Операции с каталогами

`cd`	Изменение текущего каталога
`pw`	Вывод имени текущего каталога
`basename`	Вывод конечной части пути к файлу (обычно это имя файла)
`dirname`	Вывод пути к файлу без конечной части
`mkdir`	Создание каталога
`rmdir`	Удаление каталога
`rm -r`	Удаление непустого каталога и его содержимого

О структуре каталогов Linux я рассказывал в разделе «Файловая система» главы 1. Давайте рассмотрим команды, с помощью которых можно создать, изменить или удалить каталоги.

cd

stdin **stdout** -file --opt **--help** --version

```
cd [каталог]
```

Команда `cd` (change directory) изменяет текущий каталог на указанный:

```
→ cd /usr/games
```

Если каталог не задан, то команда `cd` по умолчанию переходит в ваш домашний каталог:

```
→ cd
```

Если вместо имени каталога поставить дефис (`-`), команда `cd` возвратится в предыдущий каталог в текущей оболочке и выведет его путь:

```
→ cd /etc                    Исходное положение — /etc
→ cd /bin                    Переход в любой другой каталог
→ cd -                       Возвращение в каталог /etc
/etc
```

pwd

stdin **stdout** -file --opt **--help** --version

```
pwd
```

Команда `pwd` (print working directory) выводит абсолютный путь к текущему каталогу:

```
→ pwd
/users/smith/linuxpocketguide
```

basename

stdin **stdout** -file --opt **--help** **--version**

```
basename путь [расширение]
```

Команда `basename` выводит конечную часть *пути* к файлу. Не обязательно, чтобы путь существовал в вашей файловой системе:

```
→ basename /users/smith/finances/money.txt
money.txt
→ basename any/string/you/want                    Произвольная строка
want
```

Если вы указали опциональное *расширение*, оно будет удалено из вывода:

```
→ basename /users/smith/finances/money.txt .txt
money
```

dirname

stdin **stdout** -file --opt **--help** **--version**

```
dirname путь
```

Команда `dirname` выводит *путь* к файлу с удаленной конечной частью пути:

```
→ dirname /users/smith/mydir
/users/smith
```

`dirname` не изменяет текущий каталог вашей оболочки. Команда извлекает и выводит строку, как и `basename`.

mkdir

stdin stdout -file **--opt** **--help** **--version**

```
mkdir [параметр(ы)] каталог(и)
```

`mkdir` создает один или несколько *каталогов*:

```
→ mkdir каталог1 каталог2 каталог3
```

Полезные параметры

`-p` При задании пути к каталогу (не по имени каталога) автоматически создаются все необходимые родительские каталоги. Команда

```
→ mkdir -p one/two/three
```

создает каталоги `one`, `one/two` и `one/two/three`, если они не существуют

`-m` *режим* Создание каталога с указанными правами доступа:

```
→ mkdir -m 0755 publicdir
```

По умолчанию права доступа задаются в соответствии с настроенной в вашей оболочке пользовательской маской. См. команду `chmod` в разделах «Свойства файлов» главы 2 и «Права доступа» главы 1

---

rmdir

stdin stdout -file **--opt** **--help** **--version**

`rmdir [`*параметр(ы)*`] `*каталог(и)*

Команда `rmdir` (remove directory) удаляет пустые *каталоги*, имена которых переданы в качестве аргументов:

```
→ mkdir /tmp/junk                    Создать каталог
→ rmdir /tmp/junk                    Удалить каталог
```

Полезные параметры

`-p` Если вы указываете путь к каталогу (а не просто имя каталога), то автоматически будет удален не только нужный каталог, но и указанные родительские каталоги, которые должны быть пусты. Таким образом, команда `rmdir -p one/two/three` удаляет не только каталог `three`, но и `two`, и `one`

Чтобы удалить непустой каталог и его содержимое, выполните команду `rm -r`. Для интерактивного удаления используйте команду `rm -ri`, а для удаления целых деревьев каталогов без вывода сообщений об ошибках — `rm -rf`.

Просмотр файлов

`cat`	Просмотр содержимого файлов полностью
`less`	Просмотр текстовых файлов постранично
`nl`	Просмотр текстовых файлов с нумерацией строк
`head`	Просмотр первых строк текстового файла
`tail`	Просмотр последних строк текстового файла
`strings`	Отображение читаемого текста двоичного файла
`od`	Отображение данных в восьмеричном или других форматах

В некоторых файлах есть читаемый текст, а в других — только двоичные данные. Давайте посмотрим, как отобразить содержимое файлов разными способами.

cat

stdin stdout -file --opt --help --version

```
cat [параметр(ы)] [файл(ы)]
```

Самое простое средство просмотра — инструмент `cat`, показывающий содержимое *файлов* на стандартном выводе. Данное средство просмотра также конкатенирует их (отсюда и название команды):

```
→ cat файл          Вывод содержимого файла на экран полностью
→ cat файл*         Вывод содержимого нескольких файлов
→ cat файл* | wc    Конкатенация файлов и передача результата команде wc
```

Если в файле много строк и ваш экран не вмещает их все, используйте команду `less`, чтобы выводить содержимое постранично.

Полезные параметры

`-T`	Вывод отступов как `^I`
`-E`	Вывод переводов строк как `$`
`-v`	Вывод других непечатаемых символов в текстовом формате
`-n`	Добавление номеров ко всем строкам. (Команда `nl` намного мощнее.)
`-b`	Добавление номеров к непустым строкам
`-s`	Объединение последовательных пустых строк в одну

less

stdin stdout -file --opt --help --version

```
less [параметр(ы)] [файл(ы)]
```

Используйте команду `less`, чтобы просматривать содержимое *файла* по одной «странице» за раз (то есть по одному экрану за раз):

```
→ less файл
```

Команда отлично подходит для работы с длинными текстовыми файлами, а также может применяться в конце конвейера с длинным выводом:

```
→ команда1 | команда2 | команда3 | команда4 | less
```

Во время работы `less` введите `h`, чтобы вывести справочную информацию о команде. Вот несколько полезных клавиш для работы с командой.

Клавиша	Значение
h, H	Открытие страницы справки
Пробел, f, ^V, ^F	Перемещение вперед на один экран
Enter	Перемещение вперед на одну строку
b, ^B, ESC-v	Перемещение на один экран назад
/	Режим поиска. Введите выражение и нажмите клавишу Enter — `less` найдет и отобразит первую совпадающую строку
?	То же самое, что и /, но поиск выполняется в обратном порядке
n	Следующее совпадение — повторение последнего поиска
N	Просмотр результатов последнего поиска в обратном порядке
v	Редактирование текущего файла с помощью текстового редактора по умолчанию (значение переменной окружения `VISUAL` или, если она не определена, `EDITOR` либо системный редактор по умолчанию)
<, g	Переход к началу файла
>, G	Переход к концу файла
:n	Переход к следующему файлу
:p	Переход к предыдущему файлу

У `less` немало возможностей, но я говорю только о самых распространенных. (Во множестве дистрибутивов `less` может отображать содержимое ZIP-файлов, например, так: `less zipfile.zip`.)

Полезные параметры

`-c` Очистка экрана перед отображением следующей страницы. Позволяет избежать прокрутки, а файл просматривать удобнее

`-m` Вывод информации о проценте прокрутки файла на данный момент

`-N` Отображение номеров строк

`-r` Литеральное отображение управляющих символов

`-s` Объединение нескольких соседних пустых строк в одну строку

`-S` Обрезка длинных строк по ширине экрана

---

nl

stdin stdout -file --opt --help --version

`nl [`*`параметр(ы)`*`] [`*`файл(ы)`*`]`

Команда отправляет *файлы* на стандартный вывод с нумерацией строк:

```
→ nl poem
     1  Once upon a time, there was
     2  a little operating system named
     3  Linux, which everybody loved.
```

`nl` обеспечивает больший контроль над нумерацией, чем `cat -n`.

Полезные параметры

`-b [a|t|n|p `*`R`*`]` Нумерация всех строк (`a`), непустых строк (`t`), пустых строк (`n`) или только строк, содержащих выражение *R* (по умолчанию — `a`)

`-v `*`N`* Начало нумерации с целого числа *N* (по умолчанию — `1`)

`-i `*`N`* Увеличение нумерации каждой строки на *N*. Например, вывод только нечетных чисел (`-i2`) или только четных (`-v2 -i2`) (по умолчанию — `1`)

`-n [ln|rn|rz]` Форматирование чисел — выравнивание по левому краю (`ln`), по правому краю (`rn`) или по правому краю с ведущими нулями (`rz`) (по умолчанию — `ln`)

`-w `*`N`* Ограничение ширины числа в *N* столбцов (по умолчанию — `6`)

`-s `*`S`* Добавление строки *S* между номером строки и текстом (по умолчанию добавляется символ табуляции)

head

stdin stdout -file --opt --help --version

```
head [параметр(ы)] [файл(ы)]
```

Команда `head` выводит первые десять строк *файла*, и это очень удобно для предварительного просмотра содержимого:

```
→ head файл
→ head файл* | less          Предварительный просмотр нескольких файлов
```

Команда полезна также для предварительного просмотра результатов работы конвейера. Вывод списка из десяти последних измененных файлов в текущем каталоге:

```
→ ls -lta | head
```

Полезные параметры

`-n` *N*	Вывод первых *N* строк вместо 10
-N	То же самое, что и `-n` *N*
`-c` *N*	Вывод первых *N* байтов файла
`-q`	Режим без сообщений — при обработке более чем одного файла не выводится сообщение (с именем файла) с каждым файлом

tail

stdin stdout -file --opt --help --version

```
tail [параметр(ы)] [файл(ы)]
```

Команда `tail` выводит последние 10 строк файла:

```
→ tail файл
→ nl файл | tail             Просмотр также номеров строк
```

Полезный параметр `-f` заставляет команду следить за файлом, пока другая программа пишет в него данные. После команда `tail` отображает новые строки по мере их записи. Это очень полезно для просмотра активно используемого файла журнала Linux:

```
→ tail -f /var/log/syslog    Или другого файла журнала
```

Полезные параметры

`-n` *N*	Вывод последних *N* строк вместо 10
-N	То же самое, что и `-n` *N*
`-n` *+N*	Вывод содержимого от строки *N* до конца файла
+N	То же самое, что и `-n` *+N*
`-c` *N*	Вывод последних *N* байт файла
`-f`	Файл не закрывается, пока в него добавляются новые строки. Также они выводятся на экран. Следует добавить параметр `--retry`, если файл не существует. Сочетание клавиш ^C позволяет выйти из программы
`-q`	Режим без сообщений — при обработке более чем одного файла не выводится сообщение (с именем файла) с каждым файлом

strings

stdin **stdout** -file **--opt** **--help** **--version**

`strings [`*параметр(ы)*`] [`*файл(ы)*`]`

Двоичные файлы, например скомпилированные программы, часто содержат читаемый текст: информацию о версии, имена авторов и пути к файлам. Команда `strings` извлекает его:

```
→ strings /bin/bash
/lib64/ld-linux-x86-64.so.2
@(#)Bash version 5.1.16(1) release GNU
comparison operator expected, found '%s'
⋮
```

Комбинируйте команды `strings –n` и `grep`, чтобы сделать поиск более эффективным. Вот так можно найти адреса электронной почты:

```
→ strings -n 10 /bin/bash | grep @
bash-maintainers@gnu.org
```

Полезные параметры

`-n` *длина*	Вывод только тех строк, размер которых больше указанной *длины* (по умолчанию — 4)
`-f`	Вывод имени файла в каждой строке результата

od

stdin stdout -file --opt --help --version

od [*параметр(ы)*] [*файл(ы)*]

Чтобы просмотреть двоичные файлы, воспользуйтесь командой od. Она отображает данные файлов в ASCII, восьмеричной, десятичной или шестнадцатеричной системах счисления в разных размерах (байт, короткий, длинный). Например, следующая команда:

```
→ od -w8 /usr/bin/who
0000000 042577 043114 000402 000001
0000010 000000 000000 000000 000000
0000020 000003 000076 000001 000000
⋮
```

отображает байты в двоичном файле /usr/bin/who в восьмеричной системе счисления, по 8 байт в строке. Крайний левый столбец — это сдвиг в каждой строке в восьмеричной системе.

Если в двоичном файле также есть текст, используйте параметр -tc, который позволит вывести символьные данные. Например, двоичные файлы типа who в начале содержат строку «ELF»:

```
→ od -tc -w8 /usr/bin/who | head -3
0000000 177   E   L   F 002 001 001  \0
0000010  \0  \0  \0  \0  \0  \0  \0  \0
0000020 003  \0   >  \0 001  \0  \0  \0
```

Полезные параметры

-N *B* — Отображение первых *B* байт каждого файла, указанных в десятичной и шестнадцатеричной (с добавлением 0x) системе счисления, 512-байтовом блоке (с добавлением b), в килобайтах (с добавлением k) или мегабайтах (с добавлением m)

-j *B* — Начало вывода с байта *B* + 1 каждого файла. Допускаются те же системы счисления, что и для параметра -N (по умолчанию — 0)

-w [*B*] — Отображение *B* байт в строке. Допускаются те же системы счисления, что и для параметра -N. Параметр -w эквивалентен параметру -w32 (по умолчанию — 16)

`-s [B]`	Группировка каждого ряда байтов в последовательности из *B* байт, разделенных пробелами. Допускаются те же системы счисления, что и для параметра `-N`. Параметр `-s` эквивалентен параметру `-s3` (по умолчанию — 2)
`-A (d|o|x|n)`	Отображение сдвигов в крайнем левом столбце в десятичном (`d`), восьмеричном (`o`), шестнадцатеричном (`x`) формате или без них (`n`) (по умолчанию — `o`)
`-t(a|c)[z]`	Отображение вывода в символьном формате с неалфавитно-цифровыми символами в виде экранированной последовательности (`c`) или по имени (`a`)
`-t(d|o|u|x)[z]`	Отображение вывода в целочисленном формате: восьмеричном (`o`), десятичном с плавающей точкой (`d`), десятичном без плавающей точки (`u`), шестнадцатеричном (`x`)

Если вы добавите символ `z` к параметру `-t`, то команда выведет печатаемые символы в каждой строке в столбце справа.

Создание и редактирование файлов

`nano`	Простой текстовый редактор, доступен в большинстве дистрибутивов Linux
`emacs`	Мощный текстовый редактор компании Free Software Foundation
`vim`	Мощный текстовый редактор, основанный на Unix vi

Чтобы продуктивно и успешно использовать Linux, вы должны в совершенстве владеть одним из его текстовых редакторов. В этой ОС есть три основных текстовых редактора: nano, Emacs и Vim. Полное обучение работе с редактором выходит за рамки тематики данной книги, но в интернете можно найти множество онлайн-учебников для каждого редактора. В табл. 2.1 я приведу общие операции редакторов. Чтобы отредактировать файл, выполните одну из следующих команд:

```
→ nano файл
→ emacs файл
→ vim файл
```

Если *файл* не существует, редактор автоматически создаст его.

Таблица 2.1. Основные сочетания клавиш в текстовых редакторах

Задача	Emacs	Nano	Vim
Ввести текст	Просто вводить текст	Просто вводить текст	При необходимости перейти в режим вставки (нажмите i) и вводить текст
Сохранить и выйти	^x^s, затем ^x^c	^o, затем ^x	:wq
Выйти без сохранения	^x^c, затем выбрать «нет» при запросе о сохранении	^x, затем выбрать «нет» при запросе о сохранении	:q!
Сохранить	^x^s	^o	:w
Сохранить как	^x^w	^o, затем ввести имя файла	:w *файл*
Отменить	^/ или ^x u	M-u	u
Остановить работу редактора (не в X)	^z	^z	^z
Перейти в режим вставки	—	—	i
Перейти в режим подачи команд	—	—	ESC
Перейти в командный режим	M-x	—	:
Прервать выполнение программы	^g	^c	ESC
Переместиться вперед	^f или →	^f или →	l или →
Переместиться назад	^b или ←	^b или ←	h или ←
Переместиться вверх	^p или ↑	^p или ↑	k или ↑
Переместиться вниз	^n или ↓	^n или ↓	j или ↓
Перейти к следующему слову	M-f	^Пробел	w
Перейти к предыдущему слову	M-b	M-Пробел	b
Перейти к началу строки	^a	^a	0

Задача	Emacs	Nano	Vim
Перейти в конец строки	^e	^e	$
Переместиться на один экран вниз	^v	^v	^f
Переместиться на один экран вверх	M-v	^y	^b
Перейти в начало документа	M<-	M-\	gg
Перейти в конец документа	M->	M-/	G
Удалить следующий символ	^d	^d	x
Удалить предыдущий символ	Backspace	Backspace	X
Удалить следующее слово	M-d	—	de
Удалить предыдущее слово	M-Backspace	—	db
Удалить текущую строку	^a^k	^k	dd
Удалить до конца строки	^k	^k*	D
Определить область (нажмите клавишу, чтобы отметить начало области, а затем переместите курсор в конец области)	^Пробел	^^ (Ctrl+^)	v
Вырезать область	^w	^k	d
Копировать область	M-w	M-^	y
Вставить область	^y	^u	p
Получить справку	^h	^g	:help
Посмотреть документацию	^h i	^g	:help

* Только в том случае, если функция удаления до конца была вызвана нажатием клавиш M-k.

В Linux есть программы для редактирования документов Microsoft Office: LibreOffice (любые документы), AbiWord (только Word) и Gnumeric (только Excel). Скорее всего, некоторые из них уже включены в ваш дистрибутив, в противном случае всегда можно найти их в интернете.

Быстрое создание файла

Текстовые редакторы создают файлы по запросу, но вы можете создать пустой файл (для редактирования в будущем) в командной строке с помощью команды `touch`:

```
→ touch newfile          Создание нового файла, если его не существует
```

или используйте перенаправление вывода (см. раздел «Ввод, вывод и перенаправление» главы 1), чтобы создать файл с содержимым или пустой[1]:

```
→ echo -n > newfile2     Передача пустой строки в файл
→ > newfile3             Перенаправление в файл
→ ls > newfile4          Перенаправление вывода команды в файл
```

Ваш редактор по умолчанию

Различные программы Linux при необходимости самостоятельно запускают редактор. Например, почтовый клиент может вызвать редактор для создания нового сообщения, а программа `less` вызывает редактор в случае, если вы нажмете клавишу `v`. Обычно в качестве редактора по умолчанию задан nano или Vim. Если вы хотите выбрать другой редактор по умолчанию, настройте переменные окружения `VISUAL` и `EDITOR` по своему усмотрению, например, так:

```
→ EDITOR=emacs
→ VISUAL=emacs
→ export EDITOR VISUAL
```

Обе переменные необходимы, так как разные программы используют различные переменные. Вы можете задать их значения в файле конфигурации bash, чтобы выбранный вами вариант был зафиксирован в системе. Любая программа

[1] Параметр –n программы echo подавляет символ перевода строки.

может стать редактором по умолчанию, если принимает в качестве аргумента имя файла.

Независимо от того, как вы настроите эти переменные, я рекомендую выучить основные команды для всех трех редакторов на случай, если команда внезапно вызовет один из них на критически важном файле.

---

nano

stdin stdout -file --opt --help --version

```
nano [параметр(ы)] [файл(ы)]
```

Nano — это базовый текстовый редактор, входящий в состав большинства дистрибутивов Linux. Чтобы вызвать его для редактирования файла, выполните команду

```
→ nano файл
```

Чтобы посмотреть все клавиши и команды nano, нажмите сочетание клавиш ^G.

Обычно для выполнения большинства команд Nano нужно нажать и удерживать клавишу управления и нажать клавишу с буквой, например, ^o для сохранения и ^x для выхода. Nano удобно отображает распространенные команды в нижней части окна редактирования, хотя некоторые слова будут непонятны. (Так, термин WriteOut означает «сохранить файл».) Другие команды связаны с клавишей *meta* (обычно это Esc или Alt). В документации клавиша meta обозначается M– (например, M–F означает «нажав и удерживая клавишу meta, нажмите клавишу F»), поэтому в данной книге я использую именно эту клавишу meta. Основные сочетания клавиш представлены в табл. 2.1. Дополнительную документацию можно найти на сайте https://oreil.ly/p_Pvk.

---

emacs

stdin stdout -file --opt --help --version

```
emacs [параметр(ы)] [файл(ы)]
```

Emacs — это мощная среда редактирования с тысячами команд и богатым языком программирования для определения

собственных функций редактирования. Чтобы вызвать Emacs для редактирования файла, выполните команду

```
→ emacs файл
```

На графическом Рабочем столе откроется окно Emacs. Чтобы запустить Emacs без открытия окна, выполните команду

```
→ emacs -nw файл
```

Чтобы вызвать встроенный учебник Emacs, запустите редактор и нажмите сочетание клавиш ^h t. Большинство команд Emacs связаны с клавишей управления (например, ^F) или клавишей meta (обычно Esc или Alt). В документации клавиша meta обозначается M– (например, M–F означает «нажав и удерживая клавишу meta, нажмите клавишу F»), поэтому в данной книге я использую именно эту клавишу meta. Основные сочетания клавиш представлены в табл. 2.1.

---

vim

stdin stdout -file --opt --help --version

```
vim [параметр(ы)] [файл(ы)]
```

Vim — это улучшенная версия старого стандартного редактора vi. Чтобы вызвать редактор Vim, выполните команду

```
→ vim файл
```

Обычно Vim запускается в текущей оболочке или в окне оболочки. Чтобы запустить редактор на графическом Рабочем столе в новом окне, выполните команду

```
→ gvim файл
```

Чтобы открыть учебник по Vim из оболочки, выполните команду

```
→ vimtutor
```

Редактор работает в двух режимах: вставки и команды. В процессе редактирования вы можете переключаться между ними. *Режим вставки* используется для ввода текста обычным способом, а *режим команды* — для удаления,

копирования/вставки текста и других действий. Основные сочетания клавиш для режима команд приведены в табл. 2.1.

Свойства файлов

`stat`	Отображение атрибутов файлов и каталогов
`wc`	Подсчет байтов, слов и строк в файле
`du`	Измерение использования диска для файлов и каталогов
`file`	Идентификация типа файла
`mimetype`	Определение MIME-типа файла
`touch`	Изменение меток времени файлов и каталогов
`chown`	Изменение владельца файлов и каталогов
`chgrp`	Изменение группы пользователей для файлов и каталогов
`chmod`	Изменение режима защиты файлов и каталогов
`umask`	Настройка режима по умолчанию для новых каталогов и файлов
`lsattr`	Вывод расширенных атрибутов файлов и каталогов
`chattr`	Изменение расширенных файлов и каталогов

Изучая файлы в Linux, помните, что их содержимое — это только половина истории. Каждый файл и атрибут имеют атрибуты, описывающие их владельца, размер, права доступа и другую информацию. Команда `ls -l` (см. раздел «Основные операции с файлами» в начале данной главы) отображает некоторые из атрибутов. Другие команды позволяют просматривать и изменять эти и другие атрибуты.

---

stat

stdin **stdout** -file **--opt** **--help** **--version**

`stat [`*`параметр(ы)`*`] `*`файл(ы)`*

Команда `stat` выводит список атрибутов *файлов* (по умолчанию) или файловых систем (параметр `-f`):

```
→ stat myfile
  File: 'myfile'
```

```
  Size: 1168            Blocks: 8
  IO Block: 4096  regular file
Device: 811h/2065d      Inode: 37224455    Links: 1
Access: (0644/-rw-r--r--)  Uid: ( 600/lisa)
  Gid: ( 620/users)
Access: 2015-11-07 11:15:14.766013415 -0500
Modify: 2015-11-07 11:15:14.722012802 -0500
Change: 2015-11-07 11:15:14.722012802 -0500
 Birth: -
```

Выводится имя файла, размер в байтах (`1168`), размер в блоках (`8`), тип файла (`regular file`), права доступа в восьмеричном формате счисления (`0644`), права доступа в формате «`ls -l`» (`-rw-r--r--`), идентификатор пользователя владельца (`600`), имя владельца (`lisa`), идентификатор группы владельца (`620`), название группы владельца (`users`), тип устройства (`811` в шестнадцатеричной системе, `2065` в десятичной), номер индексного дескриптора (`37224455`), количество жестких ссылок (`1`) и метки времени последнего доступа к файлу[1], модификации, изменения состояния (прав доступа к файлу или других метаданных) и создания файла (его появления), если это возможно. Вывод `stat -f` для файловой системы выглядит следующим образом:

```
→ stat -f myfile
  File: "myfile"
    ID: f02ed2bb86590cc6 Namelen: 255
Type: ext2/ext3
Block size: 4096       Fundamental block size: 4096
Blocks: Total: 185788077  Free: 108262724
  Available: 98819461
Inodes: Total: 47202304   Free: 46442864
```

Вывод включает имя файла (`myfile`), идентификатор файловой системы (`f02ed2bb86590cc6`), максимально допустимую длину имени файла для данной файловой системы (`255` байтов), тип файловой системы (`ext`), размер блока для файловой системы (`4096`), количество общих,

[1] Большинство файловых систем больше не обновляют время доступа или делают это только при изменении файла.

свободных и доступных блоков в файловой системе (`185788077`, `108262724` и `98819461` соответственно), а также количество общих и свободных индексных дескрипторов (`47202304` и `46442864` соответственно).

Параметр `-t` показывает те же данные, но в одной строке и без заголовков. Такой формат удобен для анализа вывода:

```
→ stat -t myfile
myfile 1168 8 81a4 600 620 811 37224455 1 0 0
  1446912914 1446912914 1446912914 0 4096
→ size=$(stat -t myfile | cut -d' ' -f2)
→ echo $size                                  Второе значение
1168
```

Полезные параметры

`-L` Отслеживание символических ссылок и вывод информации о файле, на который они указывают

`-f` Вывод информации о файловой системе, в которой расположен файл

`-t` Сжатый режим — вывод информации в одну строку

WC stdin stdout -file --opt --help --version

`wc [`*`параметр(ы)`*`] [`*`файл(ы)`*`]`

Команда `wc` (word count) выводит подсчет байтов, слов и строк в текстовом *файле*:

```
→ wc myfile
  18  211 1168 myfile
```

Данный файл содержит `18` строк и `211` слов, разделенных пробелами, а его размер составляет `1168` байт.

Полезные параметры

`-l` Вывод только количества строк

`-w` Вывод только количества слов

`-c` Вывод только количества байтов

`-L` Поиск самой длинной строки в каждом файле и вывод ее длины в байтах

du

stdin **stdout** -file **--opt** **--help** **--version**

```
du [параметр(ы)] [файл(ы) | каталог(и)]
```

Команда du (disk usage) вычисляет дисковое пространство, занимаемое *файлами* и *каталогами*. По умолчанию она измеряет текущий каталог и все его подкаталоги и выводит итоговые значения в блоках для каждого подкаталога и общий итог в конце:

```
→ du
36   ./Mail
340  ./Files/mine
40   ./Files/bob
416  ./Files
216  ./PC
2404 .                                  Всего блоков
```

Команда может измерять также размер файлов:

```
→ du myfile emptyfile hugefile
4       myfile
0       emptyfile
18144   hugefile
```

Полезные параметры

`-b`	Измерение использования диска в байтах
`-k`	Измерение использования диска в килобайтах
`-m`	Измерение использования диска в мегабайтах
`-B` *N*	Отображение размера в блоках, которые определяются пользователем, где 1 блок = *N* байт (по умолчанию — `1024`)
`-h` `-H`	Вывод в удобных для восприятия единицах. Например, если два каталога имеют размер 1 Гбайт или 25 Кбайт, то `du -h` выведет `1G` и `25K` соответственно. Параметр `-h` использует значения, кратные 1024, а `-H` — значения, кратные 1000
`-L`	Отслеживание символических ссылок и измерение файлов, на которые они указывают
`-c`	Вывод итогового результата в последней строке. Стандартное действие при измерении каталога, но для измерения отдельных файлов задействуйте параметр `-c`, если вам нужно узнать общее количество
`-s`	Вывод размера каталога без учета подкаталогов

file

stdin **stdout** **-file** **--opt** **--help** **--version**

`file [`*`параметр(ы)`*`] `*`файл(ы)`*

Команда `file` сообщает тип *файла*. Она выводит предположение, основанное на содержимом файла и других факторах:

```
→ file /etc/hosts /usr/bin/who letter.docx
/etc/hosts:    ASCII text
/usr/bin/who:  ELF 64-bit LSB executable ...
letter.docx:   Microsoft Word 2007+
```

Полезные параметры

`-b`	Имена файлов не учитываются
`-i`	Вывод MIME-типов файла, например `text/plain` или `audio/mpeg`, вместо обычного вывода
`-f` *файл*	Чтение имен файлов, по одному на строку, из *файла*. Указывает типы файлов. После этого можно обрабатывать имена файлов в командной строке как обычно
`-L`	Отслеживание символических ссылок и вывод типа конечного файла вместо ссылки
`-z`	Если файл сжатый (см. раздел «Сжатие, упаковка и шифрование» далее в этой главе), то позволяет просмотреть содержимое файла без сжатия — так можно определить тип файла вместо вывода сообщения «Сжатые данные»

mimetype

stdin **stdout** **-file** **--opt** **--help** **--version**

`mimetype [`*`параметр(ы)`*`] `*`файл(ы)`*

Как и `file -i`, команда `mimetype` выводит MIME-тип *файла*, например `text/plain` или `application/pdf`, но у нее больше возможностей:

```
→ mimetype photo.jpg sample.pdf zipfile.zip
photo.jpg:  image/jpeg
sample.pdf: application/pdf
zipfile.zip: application/zip
```

Полезные параметры

`-b`	Пропуск ведущих имен файлов в выводе
`-d`	Вывод длинных описаний, например «изображение JPEG» или «архив ZIP»
`-l` *язык*	При использовании параметра `-d` вывод типов файлов на указанном *языке*. Языковой код — это стандартные двухбуквенные коды, например `ru` для русского или `cn` для китайского
`-L`	Для символических ссылок вывод типа связанного файла вместо самой ссылки (тип `inode/symlink`)

touch

stdin stdout -file **--opt** **--help** **--version**

`touch [`*`параметр(ы)`*`] `*`файл(ы)`*

Команда `touch` изменяет две метки времени, связанные с *файлом*: дату изменения (когда данные в файле были изменены в последний раз) и дату открытия (когда файл открывали в последний раз). Чтобы обновить обе метки времени, выполните команду

```
→ touch файл
```

Вы можете присвоить меткам произвольные значения, например:

```
→ touch -d "November 18 1975" файл
```

Если указанный файл не существует, команда создаст пустой файл с указанным именем (см. раздел «Быстрое создание файла» ранее в этой главе).

Полезные параметры

`-a`	Изменение только даты открытия
`-m`	Изменение только даты изменения
`-c`	Запрет создания файла, если того не существует (обычно `touch` создает файл)
`-d` *дата*	Установка метки (меток) времени. Допускается множество форматов, от `12/28/2001 3pm` до `28-May` (предполагается текущий год и время 00.00), от `next tuesday 13:59` до `0:00` (сегодняшняя полночь). Экспериментируйте и проверяйте варианты с помощью команды `stat`. Доступ к полной документации можно получить с помощью команды `info touch`

`-t` *дата* Устанавливает метку времени в формате [[*СС*]*ГГ*]*ММДДччмм*[.*сс*]]. *СС* — двузначное столетие, *ГГ* — двузначный год, *ММ* — двузначный месяц, *ДД* — двузначный день, *чч* — двузначный час, *мм* — двузначная минута и *сс* — двузначная секунда. Например, `-t 20230812150047` означает 12 августа 2023 года в 15 часов 00 минут 47 секунд

chown

stdin **stdout** **-file** **--opt** **--help** **--version**

`chown [`*параметр(ы)*`] `*владелец файл(ы)*

Команда `chown` изменяет *владельца файлов* и каталогов. В большинстве случаев она требует прав суперпользователя. Чтобы пользователь `smith` стал владельцем нескольких файлов и каталогов, выполните команду

```
→ sudo chown smith файл1 файл2 каталог
```

Параметр *владелец* может быть любым из вариантов, приведенных в таблице.

Владелец	Значение
`smith`	Существующее имя пользователя
`1001`	Числовой идентификатор пользователя
`smith:party`	Имя пользователя и название группы, разделенные двоеточием
`1001:234`	Числовые идентификаторы пользователей и групп, разделенные двоеточием
`:party`	Существующее имя группы, которому предшествует двоеточие
`--reference=`*файл*	То же значение пользователя и группы, что и в *файле*

Полезные параметры

`--dereference` Отслеживание символических ссылок и обработка файлов, на которые они ссылаются

`-R` Рекурсивная смена владельца для всего дерева каталогов

chgrp

stdin stdout -file **--opt** **--help** **--version**

```
chgrp [параметр(ы)] группа файл(ы)
```

Команда `chgrp` (change group) присваивает группу *файлам* и *каталогам*:

```
→ chgrp users файл1 файл2 каталог
```

Параметр *группа* может быть любым из следующих вариантов:

- имя группы или ее числовой идентификатор;
- `--reference=`*файл*, чтобы установить для группы того же владельца, что и для файла.

Дополнительную информацию о группах вы найдете в разделе «Управление группами» главы 3.

Полезные параметры

`--dereference`	Отслеживание символических ссылок и обработка файлов, на которые они ссылаются
`-R`	Рекурсивная смена группы для всего дерева каталогов

chmod

stdin stdout -file **--opt** **--help** **--version**

```
chmod [параметр(ы)] режим файл(ы)
```

Команда `chmod` (change mode) защищает *файлы* и каталоги от неавторизованных пользователей в системе, настраивая *режим* (права) доступа. Типичными правами доступа служат чтение, изменение и выполнение, права могут быть ограничены владельцем файла, группой и/или всеми остальными пользователями. Аргумент *режим* может принимать три формы:

- `--reference=`*файл*, устанавливает те же права доступа, что и для другого указанного файла;

- восьмеричное (основание 8) значение длиной до 4 цифр, определяющее *абсолютные* права доступа к файлу в битах (рис. 2.2). Первая слева цифра — специальная (подробности далее), а вторая, третья и четвертая указывают владельца файла, группу файла и всех пользователей соответственно;
- одна или несколько символьных строк, задающих *абсолютные* или *относительные* права (то есть относительно текущих прав доступа к файлу). Например, при `a+r` файл доступен для чтения всем пользователям.

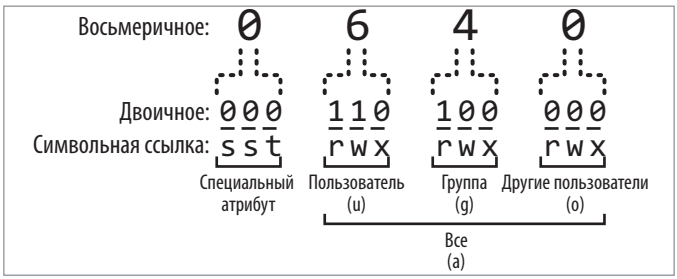


Рис. 2.2. Биты права доступа к файлу

Наиболее распространены следующие права доступа (восьмеричные значения):

```
→ chmod 600 файл      Приватный файл для вас
→ chmod 644 файл      Все могут читать, вы можете изменять
→ chmod 700 каталог   Приватный каталог для вас
→ chmod 755 каталог   Все могут читать, вы можете изменять
```

В третьей, символической форме каждая строка состоит из трех частей.

Область применения (указывать не обязательно)

`u` — пользователь, `g` — группа, `o` — другие пользователи, не входящие в группу, `a` — все пользователи. По умолчанию применяется значение `a`.

Операция

`+` — добавить права доступа, `-` — удалить права доступа, `=` — установить абсолютные права доступа, игнорируя существующие.

Права доступа

`r` — чтение, `w` — запись/изменение, `x` — выполнение (для каталога это право на вход в него), `X` — условное выполнение (объясняется позже), `u` — дублирование прав пользователя, `g` — дублирование прав группы, `o` — дублирование прав других пользователей, `s` — установка битов идентификатора пользователя или группы, `t` — липкий бит.

Например, `ug+rw` добавит права на чтение и запись для пользователя и группы, `a-x` (или просто `-x`) удалит права на изменение файла для всех, а `o=r` напрямую установит права других пользователей в режим «только чтение». Объедините эти строки, разделив их запятыми, например: `ug+rw,a-x,o=r`.

Право «условное выполнение» (`X`) означает то же самое, что и `x`, но с одним исключением: команда выполняется только в том случае, если файл представляет собой каталог или уже используется командой. В больших операциях `chmod` лучше менять бит выполнения в каталогах, но не в файлах.

Установка идентификаторов пользователя (setuid) или группы (setgid) оказывает мощное воздействие на исполняемые файлы (программы и сценарии). Предположим, что у вас есть исполняемый файл `F`, владельцем которого является пользователь `smith` и группа `friends`. Если для файла `F` установлен бит идентификатора пользователя, то любой, кто запустит файл, станет членом группы `friends` на время работы программы. Как вы можете представить, биты идентификатора пользователя или группы способны повлиять на безопасность системы, поэтому не применяйте их без необходимости. Команда `chmod +s` может сделать вашу систему уязвимой для атаки.

Липкий бит чаще всего используется для каталогов `/tmp` и удаляет файлы из них. Обычно, если у вас есть право изменения каталога, вы можете удалять файлы из каталога и перемещать их внутри него, даже если нет доступа к файлам. Для удаления файла из каталога с липким битом либо перемещения в нем требуется право на изменение файла.

Полезный параметр

`-R` Рекурсивное изменение права доступа для всего дерева каталогов

---

umask

stdin **stdout** -file **--opt** **--help** **--version**

`umask [`*`параметр(ы)`*`] [`*`маска`*`]`

Команда `umask` устанавливает или отображает режим оболочки для создания файлов и каталогов — они могут быть доступны для чтения, изменения и/или выполнения вам, вашей группе или всем пользователям.

Попробуйте задействовать *маску*:

```
→ umask                          Восьмеричный вывод
0002
→ umask -S                       Символьный вывод
u=rwx,g=rwx,o=rx
```

Команда `umask` технически осуществляет двоичные и восьмеричные вычисления, но начнем с чего-то простого. Примените маску `0022`, чтобы присвоить себе полные права на новые файлы и каталоги (у остальных пользователей будут права только на чтение и выполнение):

```
→ umask 0022
→ touch newfile && mkdir newdir
→ ls -ldG newfile newdir
-rw-r--r--  1 smith      0 Nov 11 12:25 newfile
drwxr-xr-x  2 smith   4096 Nov 11 12:25 newdir
```

Задействуйте маску `0002`, чтобы предоставить себе и своей группе полные права, а остальным пользователям — права на чтение и выполнение:

```
→ umask 0002
→ touch newfile2 && mkdir newdir2
→ ls -ldG newfile2 newdir2
-rw-rw-r--  1 smith       0 Nov 11 12:26 newfile2
drwxrwxr-x  2 smith    4096 Nov 11 12:26 newdir2
```

Используйте маску `0077`, чтобы представить себе полные права, а остальным — не предоставлять права доступа:

```
→ umask 0077
→ touch newfile3 && mkdir newdir3
→ ls -ldG newfile3 newdir3
-rw-------  1 smith       0 Nov 11 12:27 newfile3
drwx------  2 smith    4096 Nov 11 12:27 newdir3
```

А теперь техническое объяснение. `umask` определяет двоичное значение (основание 2), хотя обычно оно представлено в восьмеричном виде (основание 8). Команда определяет режим защиты по умолчанию, используя восьмеричное значение `0666` для файлов и `0777` для каталогов (с помощью двоичной операции `NOT AND`). Так, `umask 0002` дает режим защиты файлов по умолчанию `0664`:

```
0666 NOT AND 0002
= 000110110110 NOT AND 000000000010
= 000110110110     AND 111111111101
= 000110110100
= 0664
```

Аналогично для каталогов `0002` `NOT AND` `0777` дает режим по умолчанию `0775`.

lsattrt

stdin **stdout** -file **--opt** **--help** --version

```
lsattr [параметр(ы)] [файл(ы) | каталог(и)]
```

Файлы в Linux имеют дополнительные атрибуты, не ограничивающиеся правами доступа. Для *файлов* в файловой системе типа ext (ext3, ext4 и т. д.) можно вывести расширенные атрибуты с помощью команды `lsattr`. Изменить атрибуты можно с помощью команды `chattr`. Например,

следующий файл является неизменяемым (`i`) и не подлежит удалению (`u`):

```
→ lsattr attrfile
-u--i--- attrfile
```

Если файлы не указаны, `lsattr` выводит атрибуты всех файлов в текущем каталоге. Перечислю атрибуты далее.

Атрибут	Значение
`a`	Только добавление — в файл разрешено добавлять данные, но его нельзя отредактировать иным образом (только для суперпользователей)
`A`	Доступ без записи метки времени — при взаимодействии с файлом не обновляется метка времени доступа (`atime`)
`c`	Сжатие — данные прозрачно сжимаются при записи и не сжимаются при чтении
`d`	Без создания дампа — программа `dump` игнорирует этот файл при создании резервных копий (см. раздел «Резервное хранение и удаленное хранение» главы 4)
`i`	Неизменность — файл не может быть изменен или удален (только для суперпользователей)
`j`	Добавление данных в журнал (в файловых системах, поддерживающих журналирование)
`s`	Безопасное удаление — в случае удаления файл перезаписывается нулями
`S`	Синхронное обновление — изменения незамедлительно пишутся на диск
`u`	Неудаляемость — файл не может быть удален

Прежде чем использовать эту команду, прочитайте документацию для получения подробной информации. Особое внимание уделите моментам, связанным с файловой системой.

Полезные параметры

`-R` Рекурсивная обработка каталогов

`-a` Охват всех файлов, включая те, имена которых начинаются с точки

`-d` Если перечисляется каталог, его содержимое не охватывается, только сам каталог

chattr

stdin stdout -file **--opt** **--help** --version

```
chattr [параметр(ы)] [+|-|=]атрибут(ы) [файл(ы)]
```

Команда `chattr` изменяет расширенные *атрибуты файла*, те самые, которые отображает команда `lsattr`. Команда использует синтаксис, аналогичный `chmod`, — добавляет (`+`) или удаляет (`-`) атрибуты относительно или устанавливает атрибуты абсолютно (`=`):

```
→ sudo chattr +i attrfile                    Установка атрибута i
→ sudo rm attrfile                           Удаление не выполнено
rm: cannot remove 'attrfile': Operation not permitted
→ sudo chattr -i attrfile                    Сброс атрибута i
→ Sudo rm attrfile                           Удаление прошло успешно
```

Не все файловые системы поддерживают все атрибуты. Для получения подробной информации рекомендую прочитать документацию.

Полезный параметр

`-R` Рекурсивная обработка каталогов

Поиск файлов

`find`	Поиск файлов в иерархии каталогов
`xargs`	Преобразование списка файлов в список команд (и многое другое)
`locate`	Создание индекса файлов и поиск в нем строки
`which`	Поиск исполняемых файлов в пути поиска
`type`	Поиск исполняемых файлов в пути поиска (встроено в bash)
`whereis`	Поиск исполняемых файлов, документации и исходных кодов

В типичной системе Linux имеются сотни тысяч файлов, поэтому в ней есть команды для поиска нужных. Команда `find` просматривает каталоги в поиске требуемого файла. Команда `locate` работает гораздо быстрее, выполняя поиск по предварительно составленному индексу (создается

по мере необходимости). (В некоторых каталогах индекс создается по умолчанию каждую ночь.)

Чтобы найти программу в файловой системе, используйте команды `which` и `type`. Они проверяют все каталоги по пути поиска. Команда `type` встроена в оболочку bash, а `which` — это отдельная программа (обычно расположена в каталоге `/usr/bin/which`). Команда `type` намного быстрее и позволяет находить псевдонимы оболочки[1]. Команда `whereis` проверяет определенную группу путей, а не ваш путь поиска.

find

stdin **stdout** -file --opt **--help** **--version**

```
find [каталог(и)] [выражение]
```

Команда `find` выполняет поиск в одном или нескольких *каталогах* (и подкаталогах) файлов, соответствующих определенным критериям. Это очень мощная команда, принимающая более 50 параметров и имеющая, к сожалению, очень необычный синтаксис. Приведу несколько примеров простого поиска во всей файловой системе, начиная с текущего каталога (обозначен точкой):

Ищем файл `myfile`:

```
→ find . -type f -name myfile -print
./myfile
```

Команда выводит имена файлов, начинающиеся с myfile. Обратите внимание на то, что символ звездочки (*) экранирован, поэтому оболочка игнорирует его, а команде `find` она передается литерально:

```
→ find . -type f -name myfile\* -print
./myfile3
./myfile
./myfile2
```

[1] Оболочка tcsh задействует несколько трюков для поиска псевдонимов.

Вы можете также вывести имена всех каталогов:

```
→ find . -type d -print
.
./jpegexample
./dir2
./mydir
./mydir/dir
⋮
```

Полезные параметры

`-name` *шаблон* `-path` *шаблон* `-lname` *шаблон*	Если задан *шаблон* файла, выполняет поиск файлов с указанным именем (`-name`), путем (`-path`) или символической ссылкой (`lname`). Шаблон может включать символы оболочки *, ? и []. (Однако вы должны экранировать их, чтобы они игнорировались оболочкой и передавались в команду `find` литерально.) Пути относительны по отношению к дереву каталогов, в которых ведется поиск
`-iname` *шаблон* `-ipath` *шаблон* `-ilname` *шаблон*	Данные параметры аналогичны `-name`, `-path` и `-lname` соответственно, но не учитывают регистр
`-regex` *шаблон*	Путь (относительно искомого дерева каталогов) должен соответствовать указанному *шаблону*
`-type` *t*	Поиск файлов только типа *t*. Типы включают простые файлы (`f`), каталоги (`d`), символические ссылки (`l`), блочные устройства (`b`), символьные устройства (`c`), именованные пайпы (`p`) и сокеты (`s`)
`-atime` *N* `-ctime` *N* `-mtime` *N*	Определяет, когда к файлу последний раз обращались (`-atime`), последний раз файл изменяли (`-mtime`) или его статус менялся (`-ctime`), по формуле *N* · 24 часа назад. Используйте значение +*N* для условия «больше *N*» или -*N* для условия «меньше *N*»
`-amin` *N* `-cmin` *N* `-mmin` *N*	Определяет, когда к файлу последний раз обращались (`-amin`), последний раз файл изменяли (`-mmin`) или его статус менялся (`-cmin`) ровно *N* минут назад. Используйте значение +*N* для условия «больше *N*» или -*N* для условия «меньше *N*»
`-anewer` *другой_файл* `-cnewer` *другой_файл* `-newer` *другой_файл*	Определяет, когда к файлу обращались (`-anewer`), файл изменяли (`-newer`) или его статус изменялся (`-cnewer`) позже, чем у *другого_файла*

`-maxdepth` *N* `-mindepth` *N*	Охватывает файлы глубиной вложения не менее (`-mindepth`) или не более (`-maxdepth`) *N* уровней в искомом дереве каталогов
`-follow`	Разыменование символических ссылок
`-depth`	Рекурсивный поиск с самых глубоких уровней вложенности, а не с корня каталога
`-xdev`	Ограничение поиска одной файловой системой (то есть пересекать границы систем нельзя)
`-size` *N* `[bckw]`	Охват файлов размера *N*, который может быть представлен блоками (`b`), однобайтовыми символами (`c`), килобайтами (`k`) или двухбайтовыми словами (`w`). Используйте значение *+N* для условия «больше *N*» или *-N* для условия «меньше *N*»
`-empty`	Файл имеет нулевой размер и является обычным файлом или каталогом
`-user` *пользователь*	Файл принадлежит указанному *пользователю*
`-group` *группа*	Файл принадлежит данной *группе*
`-perm` *режим*	Файл имеет права, соответствующие определенному *режиму*. Используйте *-режим*, чтобы проверить, что указанные биты установлены, или *+режим*, чтобы проверить, что любой из указанных битов установлен

Следующие операторы группируют или отрицают части выражения.

выражение1 `-a` *выражение2*

И (используется по умолчанию, если два выражения указаны рядом, поэтому параметр `-a` необязателен).

выражение1 `-o` *выражение2*

ИЛИ.

`!` *выражение*
`-not` *выражение*

Отрицание выражения.

`(` *выражение* `)`

Порядок вычислений как на уроках алгебры. Сначала оценивается значение, находящееся в круглых скобках. Возможно, вам придется экранировать символы с помощью \.

выражение1 , выражение2

Оператор оценивает оба выражения и возвращает значение второго из них.

Определившись с критериями поиска, дайте задание `find` выполнить следующие действия с файлами, которые соответствуют критериям.

`-print`	Вывод пути к файлу относительно каталога поиска
`-printf` *строка*	Вывод указанной *строки*, к которой могут быть применены замены (по аналогии с C-функцией `printf()`)
`-print0`	Аналогично `-print`, но вместо разделения каждой строки вывода символом перевода строки применяется нулевой символ (ASCII 0). Используется при передаче вывода `find` в другую команду, список имен файлов может содержать пробельные символы. Конечно, принимающая команда должна быть способна читать и анализировать строки, разделенные нулем (например, `xargs -0`)
`-exec` *команда* `;`	Вызов *команды* оболочки. Экранируйте любые метасимволы оболочки, включая обязательную последнюю точку с запятой, чтобы оболочка игнорировала их. Кроме этого, символы `{}` (в кавычках или экранированные) представляют путь к найденному файлу. Полный пример: `find . -exec ls '{}' \;`
`-ok` *команда* `;`	То же, что и `-exec`, но данный параметр выводит сообщение перед каждым вызовом *команды*
`-ls`	Выполнение команды `ls -dils` для каждого файла
`-delete`	Выполнение команды `rm` для каждого файла. (Использовать с осторожностью!!!)

xargs

stdin **stdout** **-file** **--opt** **--help** **--version**

`xargs [`*параметр(ы)*`] [`*команда*`]`

Команда `xargs` одновременно самая странная и самая мощная команда оболочки. Она считывает строки текста из стандартного ввода, передает их *команде* и выполняет ее. Конечно, это может показаться неинтересным, но у команды есть несколько уникальных функций, например, для обработки списка найденных файлов. Предположим, что

вы создали файл `important`, в котором перечислены важные файлы, по одному в строке:

```
→ cat important
/home/jsmith/mail/love-letters
/usr/local/lib/critical_stuff
/etc/passwordfile2
⋮
```

`xargs` может легко обрабатывать каждый из этих файлов с помощью других команд Linux. Например, следующая команда запускает команду `ls -l` для всех перечисленных файлов:

```
→ cat important | xargs ls -l
```

Вы также можете просмотреть файлы, добавив параметр `less`:

```
→ cat important | xargs less
```

И даже удалить их с помощью `rm`:

```
→ cat important | xargs rm -f          Осторожно! Уничтожает файлы!
```

В каждом из случаев `xargs` считывает файл `important`, порождает и запускает новые команды. Но все меняется, когда список входных данных поступает не из файла, а из вывода другой команды, переданного на стандартный вывод. В частности, отличным кандидатом является команда `find`, выводящая список файлов. Например, давайте найдем в иерархии каталогов файлы, содержащие слово tomato:

```
→ find . -type f -print | xargs grep -l tomato
./findfile1
./findfile2
→ cat findfile1 findfile2
This file contains the word tomato.          Из файла findfile1
Another file containing the word tomato.     Из файла findfile2
```

Однако у команды есть один недостаток: она неправильно обрабатывает имена файлов с пробелами, например `my stuff`. Если `find` корректно выводит подобное имя файла, то `xargs` создаст некорректную команду `grep`:

```
grep -l tomato my stuff
```

Из листинга видно, что команда должна обработать два файла: `my` и `stuff`. А теперь представьте, что будет, если вы передадите в `xargs` команду `rm`! Команда удалит не те файлы! Чтобы избежать таких ситуаций, укажите командам `find` и `xargs` использовать между строками текста не пробел, а другой символ, например ноль. Команда `find -print0` завершает строки нулями, а `xargs -0` ожидает нулей:

```
→ find . -type f -print0 | xargs -0 grep -l tomato
```

Я поверхностно рассказал о команде `xargs` и надеюсь, что вы продолжите изучать ее самостоятельно. (Для безопасности вызывайте с помощью `xargs` безобидные команды, например `grep` и `ls`.)

Полезные параметры

`-n` *k* — Подача *k* строк ввода для каждой выполняемой команды. `-n1` гарантирует, что при каждом выполнении команды будет обрабатываться только одна строка ввода. В противном случае команда может передать несколько строк ввода в одну команду

`-0` — Установка в качестве конца строки ASCII-нуля, а не пробельного символа. Команда также будет считать все символы литеральными. Используйте этот параметр, когда ввод поступает от `find -print0`

xargs и подстановка команд

Если вы помните раздел «Заключение в кавычки» главы 1, то можете понять, что некоторые функции `xargs` можно выполнить подстановкой команд:

```
→ cat file_list | xargs ls -l        С xargs
→ ls -l $(cat file_list)             С $()
→ ls -l `cat file_list`              С обратными кавычками
```

Все эти команды действуют схожим образом, однако последние две могут не работать, если вывод `cat` окажется длиннее максимальной длины командной строки оболочки. `xargs` считывает неограниченный текст из стандартного ввода, но не из командной строки, так что эта команда лучше подходит для больших операций.

locate

stdin **stdout** -file **--opt** **--help** **--version**

```
locate [параметр(ы)]
```

Команда `locate` и ее партнер `updated` создают индекс (базу данных) расположения файлов с возможностью быстрого поиска[1]. Если вы планируете искать множество файлов в иерархии каталогов, которая практически не меняется, то лучше использовать команду `locate`. Чтобы найти один файл или выполнить более сложную и точную обработку найденных файлов, примените команду `find`.

Некоторые дистрибутивы автоматически и регулярно индексируют всю файловую систему (например, раз в день), так что вы можете просто запустить команду, и все заработает. Но если вам когда-нибудь понадобится самостоятельно создать индекс каталога и всех его подкаталогов (например, `~/linuxpocketguide`) и сохранить его в `/tmp/myindex`, выполните следующую команду:

```
→ updatedb -l0 -U ~/linuxpocketguide -o /tmp/myindex
```

(Обратите внимание на то, что `-l0` — это строчная буква L и ноль, а не число 10.) Затем найдите в индексе строку, например myfile:

```
→ locate -d /tmp/myindex myfile
/home/dbarrett/linuxpocketguide/myfile
/home/dbarrett/linuxpocketguide/myfile2
/home/dbarrett/linuxpocketguide/myfile3
```

У команды `updated` есть интересная дополнительная функция безопасности. Если запустить ее от имени суперпользователя, то можно создать индекс, отображающий только

[1] Моя команда locate взята из пакета под названием plocate. В некоторых системах есть старые пакеты под названием mlocate или slocate, у них немного другой синтаксис. Если у вас установлен пакет slocate, то просто запустите команду slocate вместо updateb из моих примеров.

«разрешенные» файлы (согласно правам доступа к файлам). Просто добавьте `sudo` и опустите параметр `-l0`:

```
→ sudo updatedb -U каталог -o /tmp/myindex
```

Параметры индексирования для updated

`-u`	Создание индекса из корневого каталога и далее по нисходящей
`-U` *каталог*	Создание индекса из *каталога*
`-l` (`0`|`1`)	Отключение (`0`) или включение (`1`) защиты. По умолчанию `1`
`-e` *путь(и)*	Исключение *пути* из индекса, пути разделяются пробельными символами
`-o` *файл*	Запись индекса в *файл*

Параметры поиска в определенном местоположении

`-d` *индекс*	Определяет используемый *индекс* (в моем примере это `/tmp/myindex`)
`-i`	Поиск без учета регистра
`-r` *шаблон*	Поиск имен файлов, соответствующих указанному *шаблону*

which

stdin **stdout** -file **--opt** **--help** **--version**

```
which файл
```

Команда `which` определяет местонахождение исполняемого *файла* в пути поиска оболочки. Чтобы найти команду `who`, выполните команду

```
→ which who
/usr/bin/who
```

Вы даже можете найти саму программу `which`:

```
→ which which
/usr/bin/which
```

Если несколько программ в пути поиска имеют одинаковое имя (например, `/bin/who` и `/usr/bin/who`), то программа сообщит только о первой.

type

stdin **stdout** -file **--opt** --help **--version**

```
type [параметр(ы)] команда(ы)
```

Команда `type`, как и `which`, определяет местоположение исполняемого файла в пути поиска вашей оболочки:

```
→ type grep who
grep is /bin/grep
who is /usr/bin/who
```

Команда также определяет настройки оболочки, такие как псевдонимы и встроенные команды:

```
→ type which type rm if
which is /usr/bin/which
type is a shell builtin
rm is aliased to `/bin/rm -i'
if is a shell keyword
```

Будучи встроенной, команда `type` работает быстрее, чем `which`, но доступна лишь в некоторых оболочках, например в bash.

whereis

stdin **stdout** -file --opt --help --version

```
whereis [параметр(ы)] файл(ы)
```

Команда `whereis` ищет указанные *файлы* во внутреннем списке каталогов. Она может найти исполняемые файлы, документацию и исходный код. Но есть несколько странностей, например, внутренний список каталогов может не включать нужные вам:

```
→ whereis nano
nano: /usr/bin/nano /usr/share/nano ...
```

Полезные параметры

`-b`
`-m`
`-s`
: Охват только двоичных исполняемых файлов (`-b`), man-документации (`-m`) или файлов исходного кода (`-s`)

`-B` *каталоги…* `-f`
`-M` *каталоги…* `-f`
`-S` *каталоги…* `-f`
	Поиск двоичных исполняемых файлов (`-B`), man-документации (`-M`) или файлов исходного кода (`-S`) в указанных *каталогах*. Перед выводом списка каталогов необходимо указать параметр `-f`

Работа с текстом в файлах

`grep`	Поиск в файле строк, соответствующих шаблону
`cut`	Извлечение столбцов из файла
`paste`	Добавление текста из нескольких файлов в столбцы
`column`	Организация текста в столбцы
`tr`	Преобразование одних символов в другие
`expand`	Преобразование символов табуляции в пробелы
`unexpand`	Преобразование пробелов в символы табуляции
`sort`	Сортировка строк текста по разным критериям
`uniq`	Поиск одинаковых строк в файле
`tac`	Реверсивный построчный вывод содержимого файлов
`shuf`	Случайное перемешивание строк файла (перестановка)
`tee`	Одновременная запись в файл и передача на стандартный вывод

Вероятно, самой сильной стороной Linux является работа с текстом: пользователь может преобразовать текстовый файл или стандартный ввод в нужную форму путем различных преобразований, часто в пакетном режиме. Для этого существует множество команд, но я остановлюсь на наиболее важных инструментах для преобразования текста.

---

grep

stdin stdout -file --opt --help --version

`grep [`*`параметр(ы)`*`] `*`шаблон(ы)`*` [`*`файл(ы)`*`]`

Команда `grep` — одна из самых полезных и мощных в Linux. Принцип ее работы прост: приняв один или несколько *файлов*, команда выведет все строки, соответствующие *шаблону*. Например, если файл `randomlines` содержит такие строки:

```
The quick brown fox jumped over the lazy dogs!
My very eager mother just served us nine pancakes.
Film at eleven.
```

и вы решите вывести все строки, содержащие слово pancake, то получите:

```
→ grep pancake randomlines
My very eager mother just served us nine pancakes.
```

Теперь применим шаблон для вывода строк, заканчивающихся восклицательным знаком:

```
→ grep '!$' randomlines
The quick brown fox jumped over the lazy dogs!
```

Команда принимает два типа шаблонов: *базовый* и *расширенный*. Различия в синтаксисе показаны в табл. 2.2. Шаблоны стоят времени, потраченного на их изучение. Они используются и в других мощных программах Linux, например `sed` и `perl`.

Таблица 2.2. Шаблоны для grep (простые) и egrep (расширенные)

grep	egrep	Значение
`.`	`.`	Любой отдельный символ
`[...]`	`[...]`	Любой отдельный символ из списка
`[^...]`	`[^...]`	Любой отдельный символ не из этого списка
`(...)`	`(...)`	Группировка
`|`	`|`	Или
`^`	`^`	Начало строки
`$`	`$`	Конец строки
`\<`	`\<`	Начало слова
`\>`	`\>`	Конец слова
`[:alnum:]`	`[:alnum:]`	Любой буквенно-цифровой символ
`[:alpha:]`	`[:alpha:]`	Любой буквенный символ
`[:cntrl:]`	`[:cntrl:]`	Любой управляющий символ
`[:digit:]`	`[:digit:]`	Любая цифра
`[:graph:]`	`[:graph:]`	Любой графический символ
`[:lower:]`	`[:lower:]`	Любая строчная буква
`[:print:]`	`[:print:]`	Любой печатаемый символ
`[:punct:]`	`[:punct:]`	Любой знак препинания

Таблица 2.2 (*окончание*)

grep	egrep	Значение
`[:space:]`	`[:space:]`	Любой пробельный символ
`[:upper:]`	`[:upper:]`	Любая прописная буква
`[:xdigit:]`	`[:xdigit:]`	Любая шестнадцатеричная цифра
`*`	`*`	Ноль или более совпадений шаблона
`\+`	`+`	Одно или более совпадений шаблона
`\?`	`?`	Ноль или одно совпадение шаблона
`\{`*n*`\}`	`{`*n*`}`	Ровно *n* совпадений шаблона
`\{`*n*`,\}`	`{`*n*`,}`	*n* или более совпадений шаблона
`\{`*n*`,`*m*`\}`	`{`*n*`,`*m*`}`	От *n* до *m* (включительно) совпадений шаблона, $n < m$
`\c`	`\c`	Литеральный символ `c`. Используйте `\*` для обозначения звездочки или `\\` для обозначения обратного слеша. В качестве альтернативы можно заключить символы в квадратные скобки

Полезные параметры

`-v` Вывод строк, *не* соответствующих шаблону

`-l` Вывод *имен* файлов, содержащих совпадающие строки, но не сами строки

`-L` Вывод *имен* файлов, *не* содержащих совпадающих строк

`-c` Вывод количества совпадающих строк

`-o` Вывод только совпадающих фрагментов, а не целых строк

`-n` Перед каждой совпадающей строкой указывается номер исходной строки (или с помощью `-b` выводится битовый сдвиг)

`-i` Соответствие без учета регистра

`-w` Совпадение слов целиком

`-x` Совпадение строк целиком. Отменяет команду `-w`

`-a` Анализ всех файлов как обычного текста. Иногда `grep` ошибочно принимает файл за двоичный и не выводит совпадающие строки

`-A` ***N*** После каждой совпадающей строки выводятся следующие ***N*** строк из этого файла

`-B` ***N*** После каждой совпадающей строки выводятся предыдущие ***N*** строк из этого файла

`-C` *N*	То же самое, что `-A` *N* и `-B` *N*, — выводятся *N* строк (из исходного файла), расположенные над *и* под каждой совпадающей строкой
`--color`	Выделение совпадающего текста другим цветом для лучшей читаемости
`-r`	Рекурсивный поиск всех файлов в каталоге и подкаталогах
`-R`	То же, что и `-r`, но с учетом перехода по символическим ссылкам
`-E`	Применение расширенных шаблонов. См. `egrep`
`-F`	Использование списков фиксированных строк вместо шаблонов. См. `fgrep`

egrep

stdin stdout -file --opt --help --version

`egrep` [*параметр(ы)*] *шаблон(ы)* [*файл(ы)*]

Команда `egrep` запускает `grep` с параметром `-E` для использования расширенных *шаблонов*[1].

fgrep

stdin stdout -file --opt --help --version

`fgrep` [*параметр(ы)*] [*шаблон(ы)*] [*файл(ы)*]

Команда `fgrep` запускает `grep` с параметром `–F`. Вместо шаблона она принимает список фиксированных строк, разделенных символами перевода строки. Например, если у вас есть файл словаря со строками (по слову на строку):

```
→ cat my_dictionary_file
aardvark
abbey
abbot
⋮
```

то `fgrep` ищет эти строки в одном или нескольких входных файлах:

```
→ fgrep -f my_dictionary_file story
a little aardvark who went to
visit the abbot at the abbey.
```

[1] Команды egrep и fgrep официально считаются устаревшими, однако на них основаны миллионы сценариев оболочки Linux.

Команда удобна при поиске неалфавитных символов, например * и {, так как они рассматриваются литерально как фиксированные строки, а не как шаблонные символы.

Лучше всего `fgrep` считывает фиксированные строки из файла, но может считывать их и из командной строки, если вы заключите их в кавычки. Чтобы найти в файле строки *one*, *two* и *three*, выполните команду

```
→ fgrep 'one          Обратите внимание, что я ввожу символы перевода строки
two
three' myfile
```

grep и символы конца строки

При проверке конца строки (`$`) в текстовых файлах, созданных в системах Microsoft Windows или macOS, команда `grep` может выдать странные результаты. В каждой ОС есть свой стандарт окончания строки. В Linux каждая строка заканчивается символом перевода строки (ASCII 10). В Windows текстовые строки заканчиваются символом возврата каретки (ASCII 13), за которым следует символ перевода строки. А в macOS текстовый файл может заканчиваться только символом перевода строки или только символом возврата каретки. Если `grep` не определяет концы строк, то нужно проверить наличие символов конца строки с помощью команды `cat -v`, отображающей возврат каретки как `^M` (для файлов, созданных не в Linux):

```
→ cat -v dosfile.txt
Uh-oh! This file seems to end its lines with^M
carriage returns before the newlines.^M
```

Чтобы удалить символы возврата каретки, используйте команду `tr -d`:

```
→ tr -d '\r' < dosfile.txt > /tmp/linuxfile.txt
→ cat -v /tmp/linuxfile.txt
Uh-oh! This file seems to end its lines with
carriage returns before the newlines.
```

cut

stdin stdout -file --opt --help --version

```
cut -(b|c|f)диапазон [параметр(ы)] [файл(ы)]
```

Команда `cut` извлекает столбцы текста из *файла*. Они определяются символьным сдвигом (например, 19-й символ каждой строки):

```
→ cut -c19 файл myfile
```

или битовым сдвигом (отлично от символов в случае, если в вашем языке используются многобайтовые символы):

```
→ cut -b19 файл myfile
```

или разграниченными полями (например, пятое поле в каждой строке файла `data.csv`, разделенного запятыми):

```
→ cat data.csv
one,two,three,four,five,six,seven
ONE,TWO,THREE,FOUR,FIVE,SIX,SEVEN
1,2,3,4,5,6,7
→ cut -f5 -d, data.csv
five
FIVE
5
```

Вы не ограничены выводом одного столбца: просто укажите диапазон (`3-16`), последовательность с запятыми (`3,4,5,6,8,16`) или оба варианта (`3,4,8-16`). Если при указании диапазона вы опускаете первое число (`-16`), то предполагается, что это `1` (`1-16`), если опускаете последнее (`5-`), то подразумевается последний столбец.

Полезные параметры

`-d` *C*	Символ *C* используется в качестве входного разделителя между полями для параметра `-f`. По умолчанию это символ табуляции
`--output-delimiter=`*S*	Строка *S* используется в качестве выходного разделителя между полями для параметра `-f`. По умолчанию это символ табуляции
`-s`	Сокрытие строк, которые не содержат разделителей. В противном случае они будут выводиться без изменений

paste

stdin stdout -file --opt --help --version

```
paste [параметр(ы)] [файл(ы)]
```

Команда `paste` противоположна команде `cut` — она рассматривает несколько *файлов* как вертикальные столбцы и объединяет их на стандартном выводе:

```
→ cat letters
A
B
C
→ cat numbers
1
2
3
4
5
→ paste numbers letters
1  A
2  B
3  C
4
5
→ paste letters numbers
A  1
B  2
C  3
   4
   5
```

Полезные параметры

`-d` *разделитель*
: Вывод символов указанного *разделителя* между столбцами, по умолчанию используется символ табуляции. Укажите один символ (`-d:`), который станет применяться всегда, или список символов (`-dxyz`), которые будут последовательно применяться в каждой строке (первый разделитель — x, затем y, затем z, затем x, затем y и т. д.)

`-s`
: Перегруппировка строк и столбцов вывода:

```
→ paste -s letters numbers
A B C
1 2 3 4 5
```

column

stdin **stdout** -file **--opt** **--help** **--version**

```
column [параметр(ы)] [файл(ы)]
```

Команда `column` считывает строки текста и выводит их в столбцы. По умолчанию она создает столько столбцов, сколько помещается по ширине вашего экрана, так что вывод может отличаться от данного:

```
→ seq 1 18                              Вывод 18 чисел
1
2
3
18

→ seq 1 18 | column                     Создание вертикальных столбцов
1    4    7    10   13   16
2    5    8    11   14   17
3    6    9    12   15   18

→ seq 1 18 | column -x                  Создание горизонтальных столбцов
1    2    3    4    5    6
7    8    9    10   11   12
13   14   15   16   17   18
```

`column` может преобразовать существующие столбцы в отформатированную таблицу:

```
→ cat threes                            Исходный файл
one two three                           Невыровненные столбцы
Do Re Mi
you and me
→ column -t threes                      Представление таблицы с помощью параметра -t
one  two  three                         Выровненные столбцы
Do   Re   Mi
You  and  me
→ column -t -N A,B,C threes             Добавление заголовков с помощью -N
A    B    C
one  two  three
Do   Re   Mi
You  and  me
```

Полезные параметры

`-s` *C*	Символ *C* указывается в качестве входного разделителя между столбцами
`-o` *C*	Символ *C* указывается в качестве выходного разделителя между столбцами
`-t`	Оформление вывода в виде таблицы
`-N` *заголовки*	Добавление в таблицу *заголовков* (требуется параметр `-t`). Указывается список значений, разделенных запятыми
`--json`	Вывод в формате JSON (требуются параметры `-t` и `-N`)

---

tr

stdin stdout -file --opt **--help --version**

```
tr [параметр(ы)] строка1 [строка2]
```

Команда `tr` выполняет простые преобразования одной *строки* в другую. Например, можно сделать все буквы в файле прописными:

```
→ cat wonderfulfile
This is a very wonderful file.
→ cat wonderfulfile | tr a-z A-Z
THIS IS A VERY WONDERFUL FILE.
```

или заменить все согласные буквы звездочками:

```
→ cat wonderfulfile | tr aeiouAEIOU '*'
Th*s *s * v*ry w*nd*rf*l f*l*.
```

или вообще удалить все гласные:

```
→ cat wonderfulfile | tr -d aeiouAEIOU
Ths s vry wndrfl fl.
```

Удалим все символы возврата каретки из текстового файла DOS, чтобы он был совместим с текстовыми утилитами Linux типа `grep`:

```
→ tr -d '\r' < dosfile.txt > linuxfile.txt
```

`tr` преобразует первый символ в *строке1* в первый символ в *строке2*, второй — во второй и т. д. Если длина *строки1* равна *N*, используются только первые *N* символов *строки2*. (Если *строка1* длиннее *строки2*, см. параметр `-t`.)

Наборы символов могут выглядеть следующим образом.

Форма	Значение
ABDG	Последовательность символов A, B, D, G
A-Z	Диапазон символов от A до Z
[*x***y*]	*y* повторений символа *x*
[:*класс*:]	*Классы* символов, принимаемые grep, например [:alnum:] и [:digit:]. См. табл. 2.2

tr поддерживает управляющие символы \a (^G — звуковой сигнал), \b (^H — возврат на шаг), \f (^L — пролистывание страницы), \n (^J — перевод строки), \r (^M — возврат каретки), \t (^I — горизонтальная табуляция) и \v (^K — вертикальная табуляция), принимаемые printf (см. раздел «Вывод на экран» главы 6), а также значение *nnn*, означающее символ с восьмеричным значением *nnn*.

tr отлично подходит для решения быстрых и простых задач, но для более сложных рекомендуется использовать команды sed, awk или другой язык программирования, например perl.

Полезные параметры

-d Удаление из входных данных символов *строки1*

-s Удаление смежных дубликатов (найденных в *строке1*) из входных данных. Например, tr -s aeiouAEIOU сожмет смежные дублированные гласные и превратит их в одиночные (reeeeeeally станет really)

-c Дополнение — обработка всех символов, *не* соответствующих *строке1*

-t Если *строка1* длиннее *строки2*, сделать их одинаковой длины и укоротить *строку1*. Если -t отсутствует, то последний символ *строки2* будет (скрытно) повторяться до тех пор, пока *строка2* не станет такой же длины, как *строка1*

expand

stdin stdout -file --opt --help --version

expand [*параметр(ы)*] [*файл(ы)*]
unexpand [*параметр(ы)*] [*файл(ы)*]

Команда expland преобразует символы табуляции в эквивалентное количество символов пробела, а unexpand действует

наоборот. По умолчанию табуляция равна восьми пробелам, но это можно изменить с помощью параметров:

```
→ expand tabfile > spacefile
→ unexpand spacefile > tabfile
```

Чтобы проверить, есть ли в файле пробелы или символы табуляции, используйте команду `cat -T`. Она отобразит символы табуляции как `^I`. Вы также можете применить команду `od -c`, которая отображает символы табуляции как `\t`.

Полезный параметр

`-t N` Определяет, что одной табуляции соответствуют *N* пробелов

---

sort

stdin stdout -file --opt --help --version

`sort [`*`параметр(ы)`*`] [`*`файл(ы)`*`]`

Команда `sort` печатает строки текста в алфавитном порядке или сортирует строки в соответствии с указанным вами правилом. Все файлы конкатенируются, результат сортируется и выводится на экран:

```
→ cat threeletters
def
xyz
abc
→ sort threeletters
abc
def
xyz
```

Полезные параметры

`-f` Сортировка без учета регистра

`-n` Сортировка по числу (10 идет после 9), а не по алфавиту

`-g` Другой метод сортировки чисел с другим алгоритмом, распознающим экспоненциальную запись (7.4e3 означает «7,4 умножить на 10^3», или 7400). Запустите `info sort` для получения подробной технической информации

`-u`	Уникальная сортировка — удаление повторяющихся строк. (Если используется с параметром `-c` и были найдены идентичные строки, то при проверке отсортированных файлов произойдет сбой.)
`-c`	Не сортировать, только проверить, не был ли отсортирован ввод. Если да, то ничего не выводить, в противном случае вывести сообщение об ошибке
`-b`	Игнорирование ведущих пробельных символов в строках
`-r`	Обращение вывода — сортировка от наибольшего к наименьшему
`-k` *ключ*	Учет *ключей* сортировки, описанных далее. (Используйте с `-t`, чтобы указать разделитель между ключами.)
`-t` *X*	Шаблон *X* в качестве разделителя полей для параметра `-k`

Ключ сортировки определяет часть строки, которую нужно учитывать при сортировке. Рассмотрим файл с именами и адресами:

```
→ cat people
George Washington,123 Main Street,New York
Abraham Lincoln,54 First Avenue,San Francisco
John Adams,39 Tremont Street,Boston
```

Обычная сортировка будет выводить строку Abraham Lincoln первой. Но вы можете отсортировать результат по второму значению (`-k2`) — адресу, если рассматривать каждую строку как три значения, разделенные запятыми (`-t`). Первый адрес в алфавитном порядке — это 123 Main Street:

```
→ sort -k2 -t, people
George Washington,123 Main Street,New York
John Adams,39 Tremont Street,Boston
Abraham Lincoln,54 First Avenue,San Francisco
```

Аналогичным способом можно отсортировать строки по третьему значению (`-k3`) — городу. Тогда Boston будет стоять первым в алфавитном порядке:

```
→ sort -k3 -t, people
John Adams,39 Tremont Street,Boston
George Washington,123 Main Street,New York
Abraham Lincoln,54 First Avenue,San Francisco
```

Общий синтаксис `-k F1[.C1][,F2[.C2]]` означает следующее.

Элемент	Значение	Значение по умолчанию
F1	Стартовое поле	Требуется
C1	Стартовая позиция на поле 1	1
F2	Конечное поле	Последнее поле
C2	Стартовая позиция на поле 2	1

`sort -k1.5` сортирует по первому полю, начиная с пятого символа: `sort -12.8.5` означает «от восьмого символа второго поля до первого символа пятого поля». Параметр `-t` изменяет поведение параметра `-k` таким образом, что он учитывает символы-разделители, например запятые, а не пробелы.

Чтобы задать несколько ключей, добавьте несколько параметров `-k`. Команда `sort` применяет каждый параметр от первого к последнему, как указано в командной строке.

uniq

stdin stdout -file --opt --help --version

```
uniq [параметр(ы)] [файл(ы)]
```

Команда `uniq` работает с последовательными повторяющимися строками текста. Например, у вас есть файл `letters2`:

```
→ cat letters2
a
b
b
c
b
```

Команда обнаружит и обработает любым указанным вами способом два последовательных символа `b`, но не третий символ `b`. По умолчанию команда удаляет последовательные повторения:

```
→ uniq letters2
a
b                                    Удалена одна буква b
c
b
```

`uniq` часто используется после сортировки файла:

```
→ sort letters2 | uniq
a
b
c
```

В этом случае остается только одна буква `b`, потому что команда `sort` разместила три буквы рядом, а `uniq` объединила их в одну. Чтобы подсчитать повторения строк, а не удалять их, используйте параметр `-c`:

```
→ sort letters2 | uniq -c
      1 a
      3 b
      1 c
```

Полезные параметры

`-c`	Подсчет смежных повторяющихся строк
`-i`	Операция без учета регистра
`-u`	Вывод только уникальных строк
`-d`	Вывод только повторяющихся строк
`-s` *N*	Пропуск первых *N* символов в каждой строке при обнаружении дубликатов
`-f` *N*	Пропуск первых *N* разделенных пробельными символами полей в каждой строке при обнаружении повторений
`-W` *N*	При обнаружении повторений учитываются только первые *N* символов в каждой строке. Если используется с параметром `-s` или `-f`, команда сначала игнорирует указанное количество символов или полей, а затем рассматривает следующие *N* символов

---

tac

stdin stdout -file --opt --help --version

`tac [`*параметр(ы)*`] [`*файл(ы)*`]`

Команда `tac` (`cat` в обратном порядке) выводит строки *файла* в обратном порядке:

```
→ cat lines
one
two
```

```
three
→ tac lines
three
two
one
```

Команда отлично подходит для обратного вывода строк, расположенных в хронологическом порядке.

Если в качестве аргументов заданы несколько имен файлов, `tac` обрабатывает каждый файл по очереди. Чтобы вывести в обратном порядке все строки во всех файлах вместе, сначала объедините их с помощью команды `cat` и передайте вывод в `tac`:

```
→ cat файл1 файл2 файл3 | tac
```

shuf

stdin stdout -file --opt --help --version

```
shuf [параметр(ы)] [файл(ы)]
```

Команда `shuf` в случайном порядке перемешивает все строки текста из файла или других источников.

```
→ cat lines                Исходный файл
one
two
three
→ shuf lines               Выполнение команды один раз
two
three
one
→ shuf lines               Повторный запуск команды. Вывод отличается
one
three
two
```

Или передайте строки в командной строке с помощью команды `shuf -e`:

```
→ shuf -e apple banana guava
guava
apple
banana
```

Или перемешайте числа в определенном диапазоне с помощью `shuf -i`:

```
→ shuf -i 0-3
3
1
0
2
```

Команда `shuf` подходит для извлечения случайных подмножеств строк из файла. Например, если у вас есть файл с именами и фамилиями людей, `shuf` может перемешать его и вывести указанное количество строк с помощью параметра `-n`:

```
→ cat names
Aaron
Ahmed
Ali
Ana
⋮
→ shuf -n3 names
Ying
Robert
Patricia
→ shuf -n1 names
Fatima
```

Полезные параметры

`-e`	Перемешивание строк, перечисленных в командной строке
`-i` *диапазон*	Перемешивание целых чисел из диапазона, например, 1–10
`-n` *K*	Вывод не более *K* строк
`-r`	Повторное перемешивание и вывод, чтобы получить неограниченный результат. Можно использовать вместе с параметром `-n`, чтобы ограничить вывод

---

tee

stdin stdout -file --opt --help --version

`tee [`*параметр(ы)*`] `*файл(ы)*

Как и `cat`, команда `tee` передает стандартный ввод в стандартный вывод без изменений. Команда одновременно пишет входные данные в один или несколько *файлов*. Чаще

всего `tee` используется в середине конвейера, записывая в файл промежуточные данные и передавая их следующей команде в конвейере:

```
→ who | tee original_who | sort
barrett   pts/1    Sep 22 21:15
byrnes    pts/0    Sep 15 13:51
silver       :0    Sep 23 20:44
silver    pts/2    Sep 22 21:18
```

Показанная команда записывает несортированный вывод `who` в файл `original_who` и передает тот же текст команде `sort`, которая выводит отсортированный вывод на экран:

```
→ cat original_who
silver       :0    Sep 23 20:44
byrnes    pts/0    Sep 15 13:51
barrett   pts/1    Sep 22 21:15
silver    pts/2    Sep 22 21:18
```

Полезные параметры

`-a` Добавление данных в конец файлов вместо перезаписи

`-i` Игнорирование прерываний

Более мощные манипуляции

В Linux представлены сотни других программ фильтрации, которые производят более сложные манипуляции с данными. Но мощные программы требуют тщательного изучения, что невозможно сделать в рамках небольшой книги. Вот несколько наиболее эффективных программ фильтрации, которые помогут вам начать работу.

awk

AWK — это язык сравнения шаблонов. Он сравнивает данные по шаблонам и выполняет действия на основе этих данных. Приведу несколько простых примеров обработки текстового файла `myfile`.

Вывести второе и четвертое слово в каждой строке:

```
→ awk '{print $2, $4}' myfile
```

Вывести все строки, длина которых меньше 60 символов:

```
→ awk 'length < 60 {print}' myfile
```

sed

Как и AWK, `sed` — это механизм поиска шаблонов, манипулирующий строками текста. Его синтаксис разделяют некоторые другие программы Linux, например Vim. Выведем файл, в котором все вхождения строки me будут заменены на YOU:

```
→ sed 's/me/YOU/g' myfile
```

Выводим строки файла с 3-й по 5-ю включительно:

```
→ sed -n '3,5p' myfile
```

m4

`mp4` — это макропроцессор. Он находит ключевые слова в файле и подставляет вместо них значения. Например, для данного файла:

```
→ cat substitutions
My name is NAME and I am AGE years old.
ifelse(QUOTE,yes,Learn Linux today!)
```

`mp4` выполняет замену (`-D`) для `NAME`, `AGE` и `QUOTE`:

```
→ m4 -D NAME=Sandy substitutions
My name is Sandy and I am AGE years old.

→ m4 -D NAME=Sandy -D AGE=25 substitutions
My name is Sandy and I am 25 years old.

→ m4 -D NAME=Sandy -D AGE=25 -D QUOTE=yes substitutions
My name is Sandy and I am 25 years old.
Learn Linux today!
```

Per, PHP, Python, Ruby

Если вам нужны еще более мощные инструменты, в Linux есть интерпретаторы для Perl, PHP, Python, Ruby и других полноценных языков программирования. Ссылки приведены в разделе «За пределами оболочки...» главы 6.

Сжатие, упаковка и шифрование

`tar`	Упаковка нескольких файлов в один файл
`gzip`	Сжатие файлов с помощью GNU Zip
`gunzip`	Распаковка файлов с помощью GNU Zip
`bzip2`	Сжатие файлов в формате BZip
`bunzip2`	Распаковка файлов в формате BZip
`bzcat`	Распаковка данных BZip в стандартный вывод
`compress`	Сжатие файлов с традиционным сжатием Unix
`uncompress`	Распаковка файлов с традиционным сжатием Unix
`zcat`	Распаковка данных в стандартный вывод
`zip`	Упаковка и сжатие файлов в формате Windows Zip
`unzip`	Распаковка файлов Windows Zip
`7z`	Упаковка, сжатие и распаковка файлов 7-Zip
`munpack`	Извлечение данных MIME в файлы
`mpack`	Преобразование файлов в формат MIME
`gpg`	Шифрование файлов с помощью GNU Privacy Guard (GnuPG)

Linux умеет упаковывать и сжимать файлы в различных форматах. Наиболее популярны форматы GNU Zip (`gzip`) с расширением `.gz` и BZip с расширением `.bz2`. Другие распространенные форматы включают ZIP-файлы из систем Windows (расширение `.zip`), файлы 7-Zip (расширения `.7z` и `.lzma`) и иногда классический Unix-компилятор (расширение `.Z`). Если вы встретите формат, о котором я не рассказываю, например файлы macOS SIT, ARC, Zoo, RAR и др., то больше информации найдете на сайте https://oreil.ly/vO7l8.

СОВЕТ

Несколько популярных команд были адаптированы для работы непосредственно со сжатыми файлами. Посмотрите команды zgrep, zless, zcmp и zdiff, которые работают так же, как grep, less, cmp и diff соответственно, но в качестве аргументов принимают сжатые файлы. Сравните сжатые и несжатые файлы:

```
→ zdiff sample1.gz sample2
```

Операция, связанная со сжатием, — это кодирование двоичных файлов в текст для вложений в электронную почту. В настоящее время большинство почтовых клиентов делают это автоматически, но я все же расскажу о командах `munpack` и `mpack`, которые декодируют и кодируют данные в командной строке соответственно. Наконец, расскажу о самой популярной команде шифрования файлов в Linux — `gpg` (GNU Privacy Guard).

---

tar

stdin stdout -file --opt --help --version

`tar [`*параметр(ы)*`] [`*файл(ы)*`]`

Команда `tar` упаковывает множество файлов и каталогов в один *файл* для удобства транспортировки, по желанию сжимая его. (Изначально команда предназначалась для резервного копирования файлов на локальный ленточный накопитель, ее название — это сокращение от tape archiver — ленточный архиватор.) Файлы TAR — самый распространенный формат упаковки файлов в Linux:

```
→ tar -czf myarchive.tar.gz mydir          Создание архива
→ ls -lG myarchive.tar.gz
-rw-r--r-- 1 smith 350 Nov  7 14:09 myarchive.tar.gz
→ tar -tf myarchive.tar.gz                 Список содержимого
mydir/
mydir/dir/
mydir/dir/file10
mydir/file1
mydir/file2
⋮
→ tar -xf myarchive.tar.gz                 Извлечение
```

Если в командной строке вы укажете какие-то файлы, то будут обработаны только они:

```
→ tar -xvf myarchive.tar.gz mydir/file3 mydir/file7
```

В противном случае будет обрабатываться весь архив.

Полезные параметры

`-c`	Создание архива из файлов и каталогов, перечисленных в качестве аргументов
`-r`	Добавление файлов в существующий архив
`-u`	Добавление новых/измененных файлов в существующий архив
`-A`	Добавление одного архива в конец другого. Например, `tar -A -f first.tar second.tar` добавляет содержимое `second.tar` к `first.tar`. Не может применяться к сжатым архивам
`-t`	Вывод содержимого (проверка) архива
`-x`	Извлечение файлов из архива
`-C` *каталог*	Извлечение файлов в указанный *каталог*
`-f` *файл*	Чтение архива из данного *файла* или его запись в *файл*. Обычно это файл TAR на диске (например, `myarchive.tar`), но может быть и ленточное устройство (например, `/dev/tape`)
`-d`	Сравнение архива с содержимым в файловой системе
`-z`	Применение сжатия `gzip`
`-j`	Применение сжатия `bzip2`
`-Z`	Применение сжатия Unix
`-v`	Подробный режим — вывод дополнительной информации
`-h`	Переход по символическим ссылкам, а не простое копирование
`-p`	При извлечении файлов восстановление их исходных прав доступа и владельцев

gzip

stdin stdout -file --opt --help --version

```
gzip [параметр(ы)] [файл(ы)]
gunzip [параметр(ы)] [файл(ы)]
zcat [параметр(ы)] [файл(ы)]
```

`gzip` и `gunzip` сжимают и распаковывают *файлы* в формате GNU Zip. Сжатые файлы имеют расширение `.gz`.

Примеры команд

`gzip` *файл*	Сжатие *файла* в архив *файл.gz*. Исходный *файл* удаляется
`gunzip` *файл.gz*	Распаковка архива *файл.gz* в *файл*. Исходный *файл.gz* удаляется

`gunzip -c` *файл.gz*	Распаковка архива *файл.gz* в стандартный вывод
`zcat` *файл.gz*	Распаковка архива *файл.gz* в стандартный вывод
`cat` *файл* | `gzip` | ...	Сжатие *файла* в конвейере
`cat` *файл.gz* | `gunzip`	Распаковка архива *файл.gz* из конвейера
`tar -czf` *tar-файл каталог*	Упаковка *каталога* в *tar-файл*, сжатый GZIP. Для вывода имен файлов по мере их обработки используется параметр `-v`

bzip2

stdin stdout -file --opt --help --version

```
bzip2 [параметр(ы)] [файл(ы)]
bunzip2 [параметр(ы)] [файл(ы)]
bzcat [параметр(ы)] [файл(ы)]
```

Команды `bzip2` и `bunzip2` сжимают и распаковывают *файлы* в формате Burrows-Wheeler. Сжатые файлы имеют расширение `.bz2`.

Примеры команд

`bzip2` *файл*	Сжатие *файла* в архив *файл.bz2*. Исходный *файл* удаляется
`bunzip2` *файл.bz2*	Распаковка архива *файл.bz2* в *файл*. Исходный *файл.bz2* удаляется
`bunzip2 -c` *файл.bz2*	Распаковка архива *файл.bz2* в стандартный вывод
`cat` *файл* | `bunzip2` | ...	Сжатие *файла* в конвейере
`cat` *файл.bz2* | `bunzip2`	Распаковка архива *файл.bz2* из конвейера
`bzcat` *файл.bz2*	Распаковка архива *файл.bz2* в стандартный вывод
`tar -cjf` *tar-файл каталог*	Упаковка *каталога* в *tar-файл*, сжатый BZIP. Для вывода имен файлов по мере их обработки используется параметр `-v`

compress

stdin stdout -file --opt --help --version

```
compress [параметр(ы)] [файл(ы)]
uncompress [параметр(ы)] [файл(ы)]
zcat [параметр(ы)] [файл(ы)]
```

Команды `compress` и `uncompress` сжимают и распаковывают *файлы* в классическом формате сжатия Unix. Сжатые файлы имеют расширение `.Z`.

Примеры команд

`compress` *файл*	Сжатие *файла* в архив *файл*`.Z`. Исходный *файл* удаляется
`uncompress` *файл*`.Z`	Распаковка архива *файл*`.Z` для создания *файла*. Исходный *файл*`.Z` удаляется
`uncompress -c` *файл*`.Z` `zcat` *файл*`.Z`	Распаковка архива *файл*`.Z` в стандартный вывод
`cat` *файл* `|` `compress` `|` ...	Сжатие *файла* в конвейере
`cat` *файл*`.Z` `|` `uncompress`	Распаковка архива *файл*`.Z` из конвейера
`tar -cZf` *tar-файл каталог*	Упаковка *каталога* в *tar-файл*. Для вывода имен файлов по мере их обработки используется параметр `-cvZf`

zip

stdin **stdout** -file --opt **--help** **--version**

```
zip архив.zip [параметр(ы)] [файл(ы)]
unzip [параметр(ы)] архив.zip [файл(ы)]
```

Команда `zip` упаковывает и сжимает *файлы* в формате Windows Zip, а `unzip` распаковывает эти файлы. Сжатые файлы имеют расширение `.zip`.

Примеры команд

`zip archive.zip` *файл1 файл2* ...	Упаковка *файлов*
`zip -r archive.zip` *каталог*	Рекурсивная упаковка *каталога*
`unzip -l archive.zip`	Вывод содержимого
`unzip archive.zip`	Распаковка архива

7z

stdin stdout -file **--opt** **--help** --version

```
7z [команда] [параметр(ы)] архив [(файл(ы) | каталог(и))]
```

Команда `7z`, также известная как `7-Zip`, упаковывает и сжимает *файлы* и *каталоги*. По умолчанию она создает архивные

файлы в формате LZMA с расширением `.7z`. Она поддерживает и другие методы сжатия, такие как ZIP, gzip и BZip2, но лучше использовать соответствующие команды Linux (`zip`, `gzip` и `bzip2`), которые чаще всего встречаются на Linux-машинах. 7z также извлекает файлы из различных архивов, даже из файлов Microsoft CAB. Полный список поддерживаемых форматов можно найти в документации.

Примеры команд

`7z a archive.7z` *файл1 файл2 ...*	Упаковка *файлов*
`7z a archive.7z` *каталог*	Упаковка *каталога* (так же, как и файлов)
`7z l archive.7z`	Вывод содержимого
`7z x archive.7z`	Распаковка архива

munpack

stdin **stdout** -file **--opt** --help --version

```
munpack [параметр(ы)] файл
mpack [параметр(ы)] файл(ы)
```

Современные почтовые программы отлично работают с вложениями, так что мы редко задумываемся о том, что происходит за кадром. Команды `munpack` и `mpack` работают с вложениями непосредственно в командной строке. Например, если у вас в файле `messagefile` есть сообщение электронной почты с вложенным изображением JPEG или документом PDF, то `munpack` может извлечь оба вложения в файлы:

```
→ munpack messagefile
beautiful.jpg (image/jpeg)
researchpaper.pdf (application/pdf)
```

Похожая программа `mpack` делает обратное — вставляет файлы в качестве вложений формата MIME. Давайте создадим файл `attachment.mime`, содержащий MIME-кодированное изображение `photo.jpg`:

```
→ mpack -o attachment.mime photo.jpg
Subject: My example photo
```

gpg

stdin stdout -file --opt --help --version

gpg [*параметр(ы)*] [*файл(ы)*]

Команда gpg шифрует и расшифровывает файлы, манипулирует цифровыми подписями, хранит ключи шифрования и делает многое другое. Она является частью приложения для шифрования GNU Privacy Guard (GnuPG).

Самый простой способ шифрования, симметричное шифрование, использует один и тот же пароль для шифрования и расшифровки файла (так что не потеряйте его!):

```
→ ls secret*
secret
→ gpg -c secret
Passphrase: xxxxxxxx                    Ввод желаемого пароля
Repeat: xxxxxxxx
→ ls secret*
secret secret.gpg                       Создание зашифрованного файла
```

Шифрование с открытым ключом применяется намного чаще симметричного. Для данного типа шифрования требуется пара ключей — открытый и закрытый. Сначала создайте пару ключей и в качестве открытого ключа используйте адрес своей электронной почты smith@example.com:

```
→ gpg --quick-generate-key smith@example.com \
  default default never
```

С помощью открытого ключа шифрования зашифруйте файл, создав файл secret.gpg:

```
→ gpg -e secret                 Примените открытый ключ по умолчанию
→ gpg -e -r ключ secret         Используйте подходящий открытый ключ
```

Зашифруйте и подпишите цифровой подписью файл, создав secret.gpg:

```
→ gpg -es secret
Passphrase: xxxxxxxx
```

Дешифруйте файл secret.gpg и проверьте подпись (если не будет предложено ввести кодовую фразу, значит, gpg взял ее из кэша):

```
→ rm secret                                  Удаление исходного файла
→ gpg secret.gpg                             Дешифрование файла
Passphrase: xxxxxxxx
gpg: encrypted with 4096-bit ELG key, ID 3EE49F4396C9,
created 2023-02-26 "John Smith <smith@example.com>"
⋮
Good signature from "John Smith <smith@example.com>"
→ ls secret*
secret secret.gpg                            Исходный файл в дешифрованном виде
```

Просмотр ключей в GnuPG:

```
→ gpg --list-public-keys
→ gpg --list-secret-keys
```

Полезные параметры

Команда `gpg` поддерживает около 100 параметров. Вот некоторые из них:

`-r` *имя*	Шифрование открытого ключа для получателя. В качестве *имени* может выступать идентификатор ключа, адрес электронной почты и иные части ключа
`-u` *имя*	Аутентификация в качестве пользователя с указанным *именем* из связки ключей
`-o` *файл*	Запись вывода в указанный *файл*
`-a`	Вывод в формате ASCII armor вместо OpenPGP. ASCII armor представляет собой обычный текст и подходит для вставки в сообщения электронной почты
`-q`	Тихий режим — сообщения во время работы не выводятся
`-v`	Вывод более подробных сообщений

Сравнение файлов

`diff`	Построчное сравнение двух файлов или каталогов
`comm`	Построчное сравнение двух отсортированных файлов
`cmp`	Побайтовое сравнение двух файлов
`shasum`	Вычисление контрольной суммы указанных файлов
`md5sum`	Вычисление контрольной суммы указанных файлов (небезопасно)

Вы можете сравнивать файлы Linux по крайней мере тремя способами:

- построчным сравнением (`diff`, `comm`), которое лучше всего подходит для текстовых файлов;

- побайтовым сравнением (`cmp`), которое лучше всего подходит для двоичных файлов;
- сравнением контрольных сумм (`shasum`). Избегайте старых уязвимых команд, например `sum`, `cksum` и `md5sum`, так как они используют небезопасные алгоритмы.

diff

stdin **stdout** **-file** **--opt** **--help** **--version**

```
diff [параметр(ы)] файл1 файл2
```

Команда `diff` сравнивает два *файла* построчно или два каталога пофайлово. Если различий не обнаружено, то команда не выдаст никакого результата. Для текстовых файлов `diff` создает подробный отчет обо всех различиях. Для двоичных файлов команда выводит сообщение, различаются файлы или нет.

По умолчанию формат вывода выглядит следующим образом:

```
Номера затронутых строк и тип изменений
< Соответствующий раздел файла1, если таковой имеется
---
> Соответствующий раздел файла2, если таковой имеется
```

Например, начнем с файла `fileA`:

```
Hello, this is a wonderful file.
The quick brown fox jumped over
the lazy dogs.
Goodbye for now.
```

Предположим, вы удалите первую строку, во второй строке измените brown на blue и добавите последнюю строку, создав `fileB`:

```
The quick blue fox jumped over
the lazy dogs.
Goodbye for now.
Linux r00lz!
```

Команда `diff` сообщает о двух различиях, помеченных `1,2c1` и `4a4`:

```
→ diff fileA fileB
1,2c1                    1-я и 2-я строки файла A стали 1-й строкой файла B
< Hello, this is a wonderful file.        1-я и 2-я строки файла A
< The quick brown fox jumped over
---                                       Разделитель разницы
> The quick blue fox jumped over          1-я строка файла B
4a4                                       В файл B добавлена 4-я строка
> Linux r00lz!                            Добавленная строка
```

Символы < и > слева — это стрелки, указывающие на `fileA` и `fileB` соответственно. Данный формат вывода применяется по умолчанию. Существуют и другие форматы вывода, позволяющие непосредственно передавать результат другим командам. Советую попробовать и другие варианты вывода.

Параметр	Формат вывода
`-c`	Формат контекстного сравнения, используется командой `patch` (`man patch`)
`-D` *макрос*	Формат предпроцессора C, синтаксис: `#ifdef` *макрос* `...` `#else` `...` `#endif`
`-u`	Унифицированный формат, объединяющий файлы с символом «-» для удаления и символом «+» для добавления. Применяется в `git`
`-y`	Формат смежного сравнения. Используйте `-W` для настройки ширины вывода
`-q`	Без вывода информации об изменениях просто вывести сообщение о том, различаются ли файлы

Команда `diff` умеет сравнивать каталоги. По умолчанию она сравнивает все одноименные файлы в этих каталогах, а также перечисляет файлы, присутствующие только в одном из каталогов:

```
→ diff dir1 dir2
```

Чтобы сравнить все деревья каталогов в обратном порядке, выполните команду `diff -r`. Она создает отчет (потенциально огромный) обо всех различиях:

```
→ diff -r dir1 dir2
```

Полезные параметры

`-b` Без учета пробельных символов

`-B` Без учета пустых строк

`-i` Без учета регистра

`-r` При сравнении каталогов выполняется поиск по подкаталогам

`diff` — это один из представителей семейства команд, работающих с различиями файлов. К ним относятся `diff3`, которая может сравнивать три файла за раз, и `sdiff`, которая объединяет различия между двумя файлами, создавая третий файл в соответствии с вашими инструкциями.

---

comm

stdin **stdout** **-file** **--opt** **--help** **--version**

`comm [`*`параметр(ы)`*`] `*`файл1 файл2`*

Команда `comm` сравнивает два отсортированных *`файла`* и выводит три столбца с результатами.

1. Все строки, найденные в *`файле1`*, но не обнаруженные в *`файле2`*.
2. Все строки, найденные в *`файле2`*, но не обнаруженные в *`файле1`*.
3. Все строки, найденные в обоих файлах.

Например, если файлы *`commfile1`* и *`commfile2`* содержат такие строки:

```
commfile1:          commfile2:
apple               baker
baker               charlie
charlie             dark
```

то команда выдаст вот такой вывод с тремя столбцами:

```
→ comm commfile1 commfile2
apple
                baker
                charlie
        dark
```

Полезные параметры

`-1` Без вывода столбца 1

`-2` Без вывода столбца 2

`-3` Без вывода столбца 3

`-23` Вывод строк, найденных в первом файле

`-13` Вывод строк, найденных во втором файле

`-12` Вывод строк, найденных в обоих файлах

---

cmp

stdin stdout -file --opt --help --version

`cmp [`*`параметр(ы)`*`] `*`файл1 файл2`*` [`*`сдвиг1`*` [`*`сдвиг2`*`]]`

Команда `cmp` сравнивает два *файла*. Если у обоих одинаковое содержимое, то она ничего не сообщит, в противном случае укажет местоположение первого отличия:

```
→ cmp myfile yourfile
myfile yourfile differ: byte 225, line 4
```

По умолчанию `cmp` не показывает различия, а только указывает, где они находятся. Команда подходит также для сравнения двоичных файлов, в отличие от `diff`, которая лучше всего работает с текстовыми файлами.

Обычно `cmp` начинает сравнение с начала каждого файла, но вы можете указать *`сдвиг`*, и тогда сравнение начнется с другой позиции:

```
→ cmp myfile yourfile 10 20
```

Сравнение начинается с 10-го символа файла `myfile` и 20-го символа файла `yourfile`.

Полезные параметры

`-l` Длинный вывод — побайтный вывод всех различий:

```
→ cmp -l myfile yourfile
225 167 127
```

В выводе говорится, что по сдвигу 225 (в десятичном счислении) в файле `myfile` есть строчная буква w (восьмеричное число 167), а в `yourfile` — прописная буква W (восьмеричное число 127)

`-s` Вывод без оповещения — ничего не печатать, просто возвратить соответствующий код: 0 — файлы совпадают, 1 — файлы не совпадают. (Могут использоваться другие коды, если сравнение не было выполнено.)

shasum

stdin stdout -file --opt --help --version

```
shasum -a (256|384|512|512224|512256) [параметр(ы)] файл(ы)
shasum --check файл
```

Команда `shasum` вычисляет и проверяет контрольные суммы, чтобы убедиться, что *файл* не изменился. По умолчанию она использует *небезопасный* алгоритм SHA-1. Добавьте параметр `-a 256` (или более высокое значение) для получения криптографически безопасных результатов. Командой, показанной далее, я создаю 256-битную контрольную сумму указанных файлов (64 шестнадцатеричные цифры) с помощью безопасного алгоритма SHA-256:

```
→ shasum -a 256 myfile                                    Алгоритм SHA-256
e8183aaa23aa9b74c7033cbc843041fcf1d1e9e937...  myfile
```

Вторая форма команды проверяет, совпадает ли контрольная сумма с оригинальным файлом, используя `--check`:

```
→ shasum -a 256 myfile myfile2 myfile3 > /tmp/mysum
→ cat /tmp/mysum
e8183aaa23aa9b74c7033cbc843041fcf1d1e9e937...  myfile
2254f6879ae5fdf174b3a2ebbdc7fb4fa41e0ddf4a...  myfile2
0bfa73d888300e3d4f5bc9ac302c1eb38e37499b5e...  myfile3
→ shasum --check /tmp/mysum
myfile: OK
myfile2: OK
myfile3: OK
→ echo "new data" > myfile2                        Файл myfile2 изменился
→ shasum --check /tmp/mysum
myfile: OK
myfile2: FAILED
myfile3: OK
shasum: WARNING: 1 computed checksum did NOT match
```

Вряд ли два разных файла будут иметь одинаковую контрольную сумму SHA-256, поэтому сравнение их контроль-

ных сумм — это надежный способ определить, различаются ли два потенциально одинаковых файла:

```
→ shasum -a 256 myfile | cut -c1-64 > /tmp/sum1
→ shasum -a 256 myfile2 | cut -c1-64 > /tmp/sum2
→ diff -q /tmp/sum1 /tmp/sum2
Files /tmp/sum1 and /tmp/sum2 differ
```

---

md5sum

stdin stdout -file --opt --help --version

`md5sum` *файлы* | `--check` *файл*

Команда `md5sum` вычисляет контрольные суммы с помощью *небезопасного* алгоритма MD5. Не используйте ее в повседневной работе. Команду следует выполнять так же, как и `shasum`:

```
→ md5sum myfile myfile2 myfile3 > /tmp/mysum      Создание
→ md5sum --check /tmp/mysum                       Проверка
```

Преобразование файлов в другие форматы

`pandoc`	Преобразование из одного языка разметки в другой
`hxselect`	Извлечение информации из HTML-файла
`jq`	Извлечение информации из файла JSON
`xmllint`	Проверка правильности и извлечение информации из XML-файла
`csvtool`	Извлечение информации из файла с разделенными запятой значениями (CSV)
`split`	Разделение файла на несколько файлов
`csplit`	Разделение файла на несколько файлов с помощью сложных критериев

Приходилось ли вам вручную преобразовывать текстовые файлы из одного формата в другой? Или извлекать значения из файлов CSV, JSON либо XML? Что же, вам больше не придется делать это самим! В Linux есть целый арсенал команд для преобразования файлов, которые избавят вас от подобной рутинной работы.

pandoc

stdin stdout -file --opt --help --version

pandoc [*параметр(ы)*] [*файл(ы)*]

Команда pandoc удивительна — она преобразует *файлы* многих форматов в другие форматы. Она поддерживает HTML, JSON, CSV, LaTeX, Microsoft DOCX и PPTX, DocBook XML, man-документацию, многочисленные разновидности разметки и вики-текста, а также десятки других форматов. Конечно, иногда приходится самостоятельно дорабатывать результаты, но они все равно очень хорошие[1].

Запустить pandoc очень просто: укажите входной файл и выберите выходной формат с помощью параметра `-t`. Команда выведет преобразованный результат. Посмотрите, как можно легко и просто преобразовать CSV-файл в HTML, разметку GitHub, LaTeX и JSON:

```
→ cat data.csv                                  Исходный файл
one,two,three,four,five,six,seven
ONE,TWO,THREE,FOUR,FIVE,SIX,SEVEN
1,2,3,4,5,6,7
→ pandoc -t html data.csv                       Преобразование CSV в HTML
<table>
<thead>
<tr class="header">
<th>one</th>
<th>two</th>
⋮
→ pandoc -t gfm data.csv                        Преобразование CSV в разметку
| one | two | three | four | five | six | seven |
|-----|-----|-------|------|------|-----|-------|
| ONE | TWO | THREE | FOUR | FIVE | SIX | SEVEN |
| 1   | 2   | 3     | 4    | 5    | 6   | 7     |
→ pandoc -t latex data.csv                      Преобразование CSV в LaTeX
```

[1] Я запустил команду pandoc, чтобы преобразовать третье издание этой книги, написанное в DocBook XML, в AsciiDoc для четвертого издания.

```
\begin{longtable}[]{@{}lllllll@{}}
\toprule
one & two & three & four & five & six & seven ...
⋮
→ pandoc -t json data.csv                    Преобразование CSV в JSON
{"blocks":[{"t":"Table","c":[[], ...
```

Перенаправьте результаты в файл или, что еще лучше, добавьте параметр `-o`, чтобы записать результаты в выходной файл (особенно если они в двоичном формате). Если `pandoc` может определить формат вывода по имени выходного файла, то параметр `-t` можно опустить:

```
→ pandoc -o page.pdf page.html              Преобразование HTML в PDF
```

Если `pandoc` не может определить формат входного файла, добавьте параметр `-f`:

```
→ pandoc -o page.pdf -f html page.html
```

Полезные параметры

У `pandoc` гораздо больше параметров, чем я перечислил, и команда поддерживает файлы конфигурации для удобной настройки некоторых из них. Ищите подробности в документации.

`--list-input-formats`	Перечисление всех поддерживаемых форматов ввода (на данный момент их 34)
`--list-output-formats`	Перечисление всех поддерживаемых форматов вывода (на данный момент их 57)
`-f` *формат*	Явное преобразование *из* указанного *формата*
`-t` *формат*	Явное преобразование *в* указанный *формат*
`-o` *файл*	Запись вывода в указанный *файл*
`--wrap=`*тип*	Переносы строк в выводе. Доступны следующие значения: `none` — не переносить, `auto` — перенос на 72-м символе строки и `preserve` — сохранить переносы в том же формате, что и в исходном файле
`--columns=`*N*	Выполнить перенос после *N*-го символа в строке (по умолчанию 72-го)

hxselect

stdin **stdout** -file --opt --help --version

```
hxselect [параметр(ы)] CSS-селекторы
```

Команда `hxselect` извлекает строки *CSS-селекторов* из HTML-данных. Например, извлечем все теги `div` из файла `page.html`:

```
→ cat page.html                                    Исходный файл
<html>
  <head>
  </head>
  <body>
    <div>
      This is the first div.
    </div>
    <div class="secondary">
      This is the second div.
    </div>
  </body>
</html>
→ hxselect 'div' < page.html                       Вывод одного div
<div>
      This is the first div
    </div><div class="secondary">
      This is the second div
    </div>
→ hxselect -c 'div' < page.html                    Содержимое
      This is the first div
      This is the second div
→ hxselect -c 'div.secondary' < page.html          По селектору
      This is the second div
```

Для достижения наилучших результатов сначала пропустите содержимое через команду `hxnormalize -x`, чтобы очистить HTML-код:

```
→ hxnormalize -x page.html | hxselect ...
```

Передайте исходный код веб-страницы команде `hxselect` с помощью `curl`:

```
→ curl url-адрес | hxnormalize -x | hxselect ...
```

Полезные параметры

`-c` Вывод только содержимого тегов, а не самих тегов

`-i` Сравнение строк с учетом регистра

---

jq

stdin stdout -file --opt --help --version

`jq [`*`параметр(ы)`*`] `*`фильтр`* `[`*`JSON_файлы`*`]`

Команда `jq` извлекает и обрабатывает данные в формате JSON в соответствии с указанным фильтром и выводит результаты на печать. Давайте рассмотрим несколько примеров использования этого мощного инструмента (подробнее в документации):

```
→ cat book.json                                    Исходный файл
{
  "title": "Linux Pocket Guide",
  "author": "Daniel J. Barrett",
  "publisher": {
    "name": "O'Reilly Media",
    "url": "https://oreilly.
  },
  "editions": [1, 2, 3, 4]
}
→ jq .title book.json                              Простое значение
"Linux Pocket Guide"
→ jq .title,.author book.json                      Несколько значений
"Linux Pocket Guide"
"Daniel J. Barrett"
→ jq .publisher book.json                          Объект
{
  "name": "O'Reilly Media",
  "url": "https://oreilly.com"
}
→ jq .publisher.url book.json                      Составное значение
"https://oreilly.com"
→ jq .editions book.json                           Массив данных
[
  1,
  2,
  3,
  4
]
```

```
→ jq .editions[0] book.json                    Одно значение массива
1
→ jq '.editions|length' book.json              Длина массива
4
→ jq '.editions|add' book.json                 Сумма 1 + 2 + 3 + 4
10
→ cat oneline.json                             Однострочный файл JSON
{"title":"Linux Pocket Guide","author": ...
→ jq < oneline.json                            Структурный вывод файла
{
  "title": "Linux Pocket Guide",
  "author": ...
  ⋮
```

Полезные параметры

`-f` *файл*	Чтение фильтра из файла, а не из командной строки
`-S`	Сортировка выходных данных по JSON-ключам

xmllint

stdin stdout -file --opt --help --version

`xmllint [`*параметр(ы)*`] [`*XML_файлы*`]`

Команда `xmllint` проверяет и извлекает данные XML. При передаче поддерживаемого XML-файла просто выводится его содержимое. Добавьте параметр `--noout` для подавления вывода:

```
→ xmllint good.xml
<?xml version="1.0"?>
<hello> </hello>
→ xmllint --noout good.xml
→ echo $?
0                                    Код успешного завершения
```

Некорректный XML-файл выдаст ошибку:

```
→ cat bad.xml
<?xml version="1.0"?>
<hello> </helo>                      Неправильная разметка
→ xmllint bad.xml
bad.xml:2: parser error : Opening and ending tag
mismatch: hello line 2 and helo
⋮
→ echo $?
1                                    Код ошибки
```

Предоставьте выражение XPath (https://oreil.ly/WZdhL) для извлечения данных:

```
→ cat book.xml                                    Исходный файл
<?xml version="1.0"?>
<book>
  <title>Linux Pocket Guide</title>
  <author>Daniel J. Barrett</author>
  <pub>
    <name>O'Reilly Media</name>
    <url>https://oreilly.com</url>
  </pub>
  <eds>
    <ed id="1">First edition</ed>
    <ed id="2">Second edition</ed>
    <ed id="3">Third edition</ed>
    <ed id="4">Fourth edition</ed>
  </eds>
</book>
→ xmllint --xpath '//book/title' book.xml
<title>Linux Pocket Guide</title>
→ xmllint --xpath '//book/title/text()' book.xml
Linux Pocket Guide
→ xmllint --xpath '//book/pub/url/text()' book.xml
https://oreilly.com
→ xmllint --xpath '//book/eds/ed[@id][4]' book.xml
<ed id="4">Fourth edition</ed>
→ xmllint --xpath '//book/eds/ed[@id][4]/text()' book.xml
Fourth edition
```

Полезные параметры

`--xpath` *выражение*	Поиск указанного выражения XPath в данных XML
`--format`	Форматированный вывод
`--noout`	Подавление вывода

---

csvtool

stdin stdout -file --opt --help --version

`csvtool` [*параметр(ы)*] *команда* [*аргументы*] *CSV_файлы*

Команда `csvtool` извлекает данные из файлов CVS. Она может извлекать столбцы:

```
→ cat data.csv                                    Исходный файл
one,two,three,four,five,six,seven
```

```
ONE,TWO,THREE,FOUR,FIVE,SIX,SEVEN
1,2,3,4,5,6,7
→ csvtool col 3 data.csv                    Один столбец
three
THREE
3
→ csvtool col 2,5-7 data.csv                Несколько столбцов
two,five,six,seven
TWO,FIVE,SIX,SEVEN
2,5,6,7
```

Команда может подсчитывать количество строк и столбцов:

```
→ csvtool height data.csv                   Количество строк
3
→ csvtool width data.csv                    Количество столбцов (предел)
7
```

Команда также может вставлять значения из каждой строки в более длинные строки:

```
→ csvtool format 'Third column is "%3"\n' data.csv
Third column is "three"
Third column is "THREE"
Third column is "3"
```

Может удалить первую строку, которая часто содержит заголовки:

```
→ csvtool drop 1 data.csv
ONE,TWO,THREE,FOUR,FIVE,SIX,SEVEN
1,2,3,4,5,6,7
```

Команда может выделить одно значение, например строку 2, столбец 6:

```
→ csvtool drop 1 data.csv \                 Удаление первой строки
  | csvtool head 1 - \                       Вывод первой оставшейся строки
  | csvtool col 6 -                          Извлечение строки 6
SIX
```

и многое другое. В man-документации очень мало информации о команде `csvtool`, для получения дополнительных сведений используйте команду `csvtool --help`.

Полезные параметры

`-c` *символ*	Изменение разделителя во вводе с запятой на указанный *символ*. Для символов табуляции используйте `-c TAB`
`-o`	То же, что и `-c`, но для символа разделителя вывода

---

split

stdin stdout -file --opt --help --version

`split [`*параметр(ы)*`] [`*файл* `[`*префикс*`]]`

Команда `split` делит *файл* на части одинакового размера и сохраняет их в отдельных файлах. По умолчанию выходные файлы именуются последовательно, например: `Xaa`, `Xab`, `Xac` и т. д. `X` служит *префиксом*, а окончания `aa`, `ab` и т. п. — *суффиксом*.

Разделим входной файл по количеству строк:

```
→ split hugefile                    1000 строк каждый
→ split -l 2000 hugefile            2000 строк каждый
```

или по байтам:

```
→ split -b 1024 hugefile            1024 байта каждый
```

или по определенному символу, называемому разделителем:

```
→ split -t@ -l1 hugefile            Разделитель @
```

Вы можете изменить префикс с помощью последнего аргумента:

```
→ split hugefile part-              Использование префикса part-
→ ls
Hugefile  part-aa  part-ab  part-ac  part-ad  ...
```

`split` поддерживает работу с очень большими файлами, которые невозможно загрузить в другие программы или нельзя хранить на носителях. Вы можете собрать файл воедино с помощью программы `cat`:

```
→ cat part-* > /tmp/outfile         Объединение файлов part-aa, part-ab и т. д.
```

Полезные параметры

`-l` *N*	Разбиение на файлы по *N* строк или записей
`-b` *N*	Разбиение на файлы размером *N* байт
`-t` *разделитель*	Использование символа-*разделителя* между записями
`-a` *длина*	Ограничение *длины* суффикса в символах
`-d`	Использование вместо букв числовых суффиксов, начиная с нуля. Начальное значение меняется на *N* с помощью `--numericsuffixes` *N*

csplit

stdin **stdout** **-file** **--opt** **--help** **--version**

```
csplit [параметр(ы)] файл шаблон
```

Команда `csplit` разделяет один файл на множество *файлов* по *шаблону*. Выходные файлы именуются последовательно, например: `xx01`, `xx02`, `xx03` и т. д. Значение `xx` (по умолчанию) называется *префиксом*, а окончания `01`, `02` и т. п. — *суффиксом*. По умолчанию `csplit` выводит размер каждого создаваемого файла.

Синтаксис команды немного непривычный. После имени входного файла укажите один или несколько шаблонов, настраивающих разделение строк в файле. Шаблоны бывают двух типов.

Регулярные выражения

Выражение (см. табл. 2.2), заключенное в слеши, например, `/^[A-Z]*$/` для строки, состоящей из прописных букв. По умолчанию каждое выражение сопоставляется однократно.

Повторитель /ретранслятор

Повторитель имеет форму `{`*N*`}`, что означает «совпадение с шаблоном до *N* раз». Примером может служить `{2}`, говорящий о том, что совпадение с шаблоном выявлено дважды. Специальный повторитель `{*}` означает совпадение с шаблоном столько раз, сколько возможно, пока не закончится ввод.

Команда `csplit` отлично подходит для разделения структурированного текста, например HTML, XML или исходного кода программы:

```
→ cat page.html                    Исходный файл
<html>
  <head>
  </head>
  <body>
    <div>
      This is the first div.
    </div>
    <div class="secondary">
      This is the second div.
    </div>
  </body>
</html>
```

Можно применить четыре шаблона (в листинге — в кавычках), чтобы разделить файл на разделы `<head>`, затем `<body>`, а затем `<div>`:

```
→ csplit page.html '/<head>/' '/<body>/' '/<div/' '{*}'
7
19
9                          Размер выходных файлов в байтах
49
86
```

Взглянем на пять выходных файлов:

```
→ ls xx*
xx00  xx01  xx02  xx03  xx04
→ cat xx00
<html>
→ cat xx01
  <head>
  </head>
→ cat xx02
  <body>
→ cat xx03
    <div>
      This is the first div
    </div>
→ cat xx04
```

```
    <div class="secondary">
      This is the second div
    </div>
  </body>
</html>
```

Полезные параметры

`-f` *префикс*	Настройка *префикса* для имен выходных файлов
`-n` *длина*	Настройка числового суффикса указанной *длины* для имен выходных файлов
`-s`	Без оповещений — размер сгенерированных файлов не выводится

Работа с файлами PDF и PostScript

`pdftotext`	Извлечение текста из файлов PDF
`ps2ascii`	Извлечение текста из файлов PostScript или PDF
`pdfseparate`	Извлечение отдельных страниц из файла PDF
`pdftk`	Разделение, объединение, поворот и другие манипуляции с файлами PDF
`pdf2ps`, `ps2pdf`	Преобразование между форматами файлов PDF и PostScript
`ocrmypdf`	Оптическое распознавание символов в файлах PDF

Для просмотра PDF-файлов и файлов PostScript вам понадобится графическая рабочая среда и программа для просмотра документов, например[1]:

```
→ okular sample.pdf      Средство просмотра документов KDE
→ evince sample.pdf      Средство просмотра документов GNOME
→ gv sample.pdf          Средство просмотра документов
```

Кроме того, в Linux есть богатый набор инструментов командной строки для работы с файлами PDF и PostScript. Давайте изучим эти инструменты (особое внимание рекомендую уделить команде `pdftk`).

---

[1] Лучшей программой для редактирования PDF-файлов является Master PDF Editor компании Code Industry (https://oreil.ly/l5Hto). Это коммерческий продукт.

pdftotext

stdin **stdout** **-file** **--opt** **--help** **--version**

```
pdftotext [параметр(ы)] [файл.pdf [файл.txt]]
```

Команда `pdftotext` извлекает текст из *файла*`.pdf` и записывает его в *файл*`.txt`. Она работает только в том случае, если PDF-текст содержит истинный текст, а не изображения с текстом (в данном случае сначала запустите команду `ocrmypdf`):

```
→ pdftotext sample.pdf                    Создание файла sample.txt
```

Полезные параметры

`-f` *N*	Начало со страницы *N*. Между параметром и номером должен быть пробел
`-l` *N*	Завершение на странице *N*. Между параметром и номером должен быть пробел
`-htmlmeta`	Генерация HTML-документа, а не текстового (создается файл `sample.html`)
`-eol` *ОС*	Запись символов конца строки для указанной операционной системы (*ОС*) — `dos`, `mac` или `unix`

ps2ascii

stdin **stdout** **-file** **--opt** **--help** **--version**

```
ps2ascii файл.(ps|pdf) [файл.txt]
```

Команда `ps2ascii` извлекает текст из *файла* PostScript. Это простая команда без параметров. Чтобы извлечь текст из файла `sample.ps` и поместить его в файл `/tmp/extracted.txt`:, выполните команду

```
→ ps2ascii sample.ps /tmp/extracted.txt
```

Команда также может извлекать текст из файла PDF:

```
→ ps2ascii sample.pdf /tmp/extracted2.txt
```

pdfseparate

stdin stdout -file **--opt** **--help** --version

```
pdfseparate [параметр(ы)] [файл.pdf] [шаблон]
```

Команда `pdfseparate` разбивает *`файл.pdf`* на несколько отдельных файлов PDF (по файлу на каждую страницу). Например, если в файле `one.pdf` есть 10 страниц, то данная команда создаст 10 файлов PDF с именами `split1.pdf` — `split10.pdf`, каждый из которых содержит одну страницу:

```
→ pdfseparate one.pdf /tmp/split%d.pdf
```

Последний аргумент — это *`шаблон`* для подстановки имен отдельных файлов страниц. Специальное обозначение `%d` означает номер извлеченной страницы.

Полезные параметры

`-f` *N* Начало работы с PDF-файлом со страницы *N*

`-l` *N* Завершение работы с PDF-файлом на странице *N*. Между параметром и номером должен быть пробел

pdftk

stdin **stdout** -file --opt **--help** **--version**

```
pdftk [аргумент(ы)]
```

`pdftk` — это многофункциональная команда для работы с PDF-файлами. Программа позволяет извлекать страницы из PDF-файла, объединять несколько PDF-файлов в один, поворачивать страницы, добавлять водяные знаки, шифровать и дешифровать файлы и многое другое... И все это делается из командной строки! К сожалению, идеальных программ нет — в этой придется привыкать к причудливому синтаксису. Но если вы потратите немного сил и времени, то сможете освоить несколько полезных трюков.

Чтобы объединить файлы `one.pdf` и `two.pdf` в один PDF-файл, `combined.pdf`, выполните команду

```
→ pdftk one.pdf two.pdf cat output combined.pdf
```

Чтобы извлечь из файла `one.pdf` страницы 3, 5 и 8–10 и записать их в файл `new.pdf`, выполните команду

```
→ pdftk one.pdf cat 3 5 8-10 output new.pdf
```

Чтобы извлечь первые пять страниц из файла `one.pdf`, нечетные страницы из файла `two.pdf` и объединить их в файле `combined.pdf`, выполните команду

```
→ pdftk A=one.pdf B=two.pdf cat A1-5 Bodd output \
  combined.pdf
```

Чтобы скопировать файл `one.pdf` в `new.pdf`, но так, чтобы страница 7 была повернута на 90° по часовой стрелке, выполните команду

```
→ pdftk one.pdf cat 1-6 7east 8-end output new.pdf
```

Чтобы чередовать страницы файлов `one.pdf` и `two.pdf` в файле `mixed.pdf`, выполните команду

```
→ pdftk one.pdf two.pdf shuffle output mixed.pdf
```

Критерии выбора страниц обычно указываются перед ключевым словом `output`. Они состоят из одного или нескольких диапазонов страниц с квалификаторами. Диапазон страниц может быть разным: одна страница (`5`), несколько страниц в прямом (`5-10`) или обратном (`10-5`) порядке. Квалификаторы могут удалять страницы из диапазона. Например, `1-100~20-25` означает: «все страницы с 1-й по 100-ю, кроме страниц с 20-й по 25-ю». С помощью ключевых слов `odd` или `even` можно указывать только четные или нечетные страницы соответственно. Поворот страницы можно задать с помощью ключевых слов `north`, `south`, `east` и `west`. Я рассказал лишь о некоторых возможностях команды `pdftk`. В документации вы найдете больше подробностей и примеров.

pdf2ps

stdin stdout **-file** --opt **--help** --version

```
pdf2ps [параметр(ы)] файл.pdf [файл.ps]
ps2pdf [параметр(ы)] файл.ps [файл.pdf]
```

Команда `pdf2ps` преобразует *файл* Adobe PDF в *файл* PostScript. Если вы не указали имя выходного файла, то

по умолчанию будет использоваться имя входного файла, при этом `.pdf` заменится на `.ps`:

```
→ pdf2ps sample.pdf converted.ps
```

Чтобы выполнить обратное действие, например преобразовать PostScript-файл в формат PDF, задействуйте команду `ps2pdf`:

```
→ ps2pdf sample.ps converted.pdf
```

---

ocrmypdf

stdin **stdout** **-file** **--opt** **--help** **--version**

`ocrmypdf [`*`параметр(ы)`*`] `*`источник назначение`*

Команда `ocrmypdf` выполняет оптическое распознавание символов для создания PDF-файла с возможностью поиска. *Источником* может быть графический файл или PDF-файл с изображением.

```
→ ocrmypdf imageoftext.png outfile.pdf      Конвертирование
→ okular outfile.pdf                        Отображение
```

Полезные параметры

`-l` *язык* — Указание *языка*, отличного от английского. Чтобы вывести список языков, выполните команду

```
tesseract --list-langs
```

`-r` — Поворот страницы до правильной ориентации

Печать

`lpr` — Печать файла

`lpq` — Просмотр очереди печати

`lprm` — Удаление задания из очереди печати

В Linux есть две популярные системы печати, CUPS и LPRng. Обе они используют команды `lpr`, `lpq` и `lprm`, но их параметры в разных системах различаются. Я расскажу вам об общих параметрах, которые поддерживаются в обеих программах.

Для установки и настройки принтеров в GNOME и KDE есть специальные интерфейсы в системных настройках. Для устранения неполадок с принтером CUPS посетите

страницу http://localhost:631 — там вы получите доступ к системе управления CUPS на своем компьютере.

lpr

stdin stdout -file --opt **--help** --version

```
lpr [параметр(ы)] [файл(ы)]
```

Команда `lpr` отправляет *файл* на принтер:

```
→ lpr файл                    Печать на принтере по умолчанию
→ lpr -P принтер файл         Печать на указанном принтере
```

Полезные параметры

`-P` *принтер*	Передача файла на настроенный *принтер*
`-#` *N*	Печать *N* копий файла. Имя параметра представляет собой литеральный символ #
`-J` *имя*	Указание *имени* задания, которое будет напечатано на титульном листе (если в системе настроена печать титульных листов)

lpq

stdin **stdout** -file --opt **--help** --version

```
lpq [параметр(ы)]
```

Команда `lpq` выводит список заданий, ожидающих печати.

Полезные параметры

`-P` *принтер*	Список очередей для *принтера*
`-a`	Список очередей для всех принтеров
`-l`	Если в двух словах — отображение информации в более длинном формате

lprm

stdin **stdout** -file --opt --help --version

```
lprm [параметр(ы)][идентификатор(ы)]
```

Команда `lprm` отменяет одно или несколько заданий. Чтобы узнать *идентификаторы* нужных заданий печати (например, `61` и `78`), выполните команду `lpq`:

```
→ lprm -P принтер 61 78
```

Если вы не указали идентификатор задания, то будет отменено текущее задание печати. (Суперпользователь может отменять задания других пользователей.) Параметр `-P` позволяет определить, в какой очереди печати находится задание.

Проверка орфографии

`look`	Быстрый поиск корректного написания слова
`aspell`	Интерактивная проверка орфографии
`spell`	Пакетная проверка орфографии

В Linux есть несколько встроенных программ для проверки орфографии. Если вы привыкли к графическим программам проверки орфографии, то текстовые программы Linux покажутся вам довольно примитивными.

look

stdin **stdout** -file **--opt** **--help** **--version**

```
look [параметр(ы)] префикс [словарь]
```

Команда `look` выводит все слова, начинающиеся с определенного строкового *префикса*. Все слова находятся в файле *словаря* (по умолчанию словарь расположен в файле `/usr/share/dict/words`):

```
→ look bigg
bigger
biggest
Biggs
```

Если вы задействуете собственный файл словаря — текстовый файл, где слова отсортированы по алфавиту, то программа `look` выведет все строки словаря, начинающиеся с указанного *префикса*.

Полезные параметры

`-f`	Игнорирование регистра. Необходимо использовать только в том случае, если вы работаете с собственным файлом словаря

aspell

stdin stdout -file --opt --help --version

```
spell [параметр(ы)] файл
```

Команда `aspell` — это интерактивная проверка орфографии. Она ищет слова, которые невозможно распознать, и предлагает альтернативные варианты. Вот несколько полезных примеров применения.

`aspell -c` *файл*

Интерактивная проверка и исправление написания всех слов в файле.

`aspell dump master`

Передача основного словаря `aspell` на стандартный вывод.

`aspell help`

Вывод краткого справочного сообщения. Подробнее об этом рассказывается на веб-сайте http://aspell.net.

spell

stdin stdout -file --opt --help --version

```
spell [файл(ы)]
```

Команда `spell` выводит все слова с ошибками в выбранных *файлах* (в соответствии с выбранным словарем). Эта команда не является интерактивной:

```
→ cat badwords
This Linux file has some spelling errors.
You may naturaly wonder if a spelling checker
will pick them up. Careful Linuxx users should
run thier favorite spelling checker on this file.
→ spell badwords
naturaly
Linuxx
thier
```

ГЛАВА 3

Основы системного администрирования

Привилегии суперпользователя

Обычные пользователи могут изменять только принадлежащие им файлы. Но один специальный пользователь — *суперпользователь*, *администратор* или *root* — имеет полный доступ к машине и может делать все что угодно. Такие привилегии предназначены для решения задач системного администрирования. Задействуйте права суперпользователя лишь при крайней необходимости, чтобы не навредить системе Linux. Входите в систему от имени суперпользователя, только если есть значимый повод (например, при восстановлении нарушенного процесса загрузки).

ВНИМАНИЕ!

Команды суперпользователя следует применять с осторожностью: они могут вывести из строя систему Linux.

Стать суперпользователем можно несколькими способами. Первый — задействовать команду `sudo`, тогда вы получите права суперпользователя на время выполнения одной команды. Просто введите слово sudo, а затем — команду. Возможно, вам придется ввести пароль (это зависит от настроек команды `sudo` на вашей машине):

```
→ sudo rm некий_защищенный_файл
[sudo] password: xxxxxxxx                    Ваш пароль
```

Чтобы сохранить права суперпользователя, запустите оболочку суперпользователя с помощью одной из следующих команд:

```
→ sudo -s
→ sudo bash
```

Оболочка, запущенная от имени суперпользователя, подходит для просмотра множества защищенных каталогов с помощью команды `cd`. Выполнив команды от имени суперпользователя, нажмите сочетание клавиш `^D` или введите `exit`, чтобы завершить работу оболочки суперпользователя. Если вы забыли, в какой оболочке работаете, проверьте профиль с помощью команды `whoami`. Если вы являетесь суперпользователем, то команда отобразит имя `root`.

Стать суперпользователем можно с помощью команды `su`, создающей оболочку суперпользователя. Но для этого вам понадобится другой пароль, называемый паролем суперпользователя (он создается во время установки Linux):

```
→ su
Password: xxxxxxxx                    Пароль суперпользователя
#
```

Если вы являетесь суперпользователем, то очень часто приглашение оболочки заменяется на октоторп (`#`). Добавьте параметр `-l`, чтобы запустить оболочку входа в систему, включающую все окружение суперпользователя, например псевдонимы оболочки суперпользователя:

```
→ su -l                               Запуск оболочки для входа в систему
Password: xxxxxxxx                    Пароль суперпользователя
#
```

Если в `sudo` или `su` вы укажете имя пользователя, то станете этим пользователем независимо от того, есть ли у вас все необходимые права:

```
→ sudo -u sophia команда              Аутентификация от имени пользователя
                                      sophia с помощью sudo
[sudo] password: xxxxxxxx             Получение доступа sudo
→ su sophia                           Аутентификация от имени пользователя
                                      sophia с помощью su
Password: xxxxxxxx                    Ввод пароля пользователя sophia
```

Рекомендую задействовать команду `sudo`, а не `su`, особенно если в вашей системе зарегистрировано несколько пользователей. Команда `su` применяет общий пароль — и это проблема для безопасности всей системы. Команда `sudo` использует ваш собственный пароль, но для этого его нужно настроить (во многих каталогах `sudo` уже настроена). `sudo` также обеспечивает контроль над правами в файле `/etc/sudoers` и даже записывает в журнал команды, которые выполняют пользователи. Полное обсуждение этого вопроса выходит за рамки тематики данной книги, ищите подробности с помощью команды `man sudo`.

Просмотр процессов

`ps`	Перечисление процессов
`pgrep`	Перечисление идентификаторов процессов, соответствующих шаблону
`uptime`	Просмотр загрузки системы
`w`	Перечисление активных процессов для всех пользователей
`top`	Мониторинг ресурсоемких процессов в интерактивном режиме
`free`	Отображение свободных ресурсов памяти

Процесс — это единица работы в системе Linux. Каждая запущенная программа представляет собой один или несколько процессов. В Linux есть команды для просмотра процессов и управления ими. Каждый процесс идентифицируется числовым *идентификатором процесса* (PID). Вы можете найти больше информации в каталоге `/proc` (см. раздел «Каталоги, связанные с ядром» главы 1).

Процессы отличаются от заданий (см. раздел «Управление заданиями оболочки» главы 1). Процессы являются частью ОС. Задания — это конструкции более высокого уровня, с которыми может взаимодействовать только оболочка. Работающая программа состоит из одного или нескольких процессов, задание оболочки состоит из одной или нескольких программ, выполняемых как команда оболочки.

ps

stdin **stdout** -file --opt **--help** **--version**

```
ps [параметр(ы)]
```

Команда `ps` отображает информацию о запущенных вами процессах, а также о процессах других пользователей (по желанию):

```
→ ps
  PID TTY          TIME CMD
 4706 pts/2    00:00:01 bash
15007 pts/2    00:00:00 emacs
16729 pts/2    00:00:00 ps
```

Команда `ps` поддерживает не менее 80 параметров, я рассмотрю лишь несколько полезных комбинаций. Может показаться, что некоторые параметры здесь не к месту и непоследовательны, поскольку команда `ps` для обеспечения совместимости включает в себя функции некоторых других команд.

Выведем запущенные процессы:

```
→ ps -ux
```

Выведем все процессы, запущенные пользователем `smith`:

```
→ ps -U smith
```

Выведем все процессы запущенной программы:

```
→ ps -C python
```

Просмотр процессов на терминале *N*:

```
→ ps -tN
```

Выведем конкретные процессы 1, 2 и 3505:

```
→ ps -p1,2,3505
```

Выведем все процессы с командными строками с ограничением по ширине экрана:

```
→ ps -ef
```

Выведем все процессы с полной командной строкой:

```
→ ps -efww
```

Выведем все процессы в потоке с отображением дочерних процессов под родительскими:

```
→ ps -efH
```

Можно добавлять в конвейер `grep` и другие команды для более точного извлечения полезной информации из вывода `ps`:

```
→ ps -ux | grep python
```

---

pgrep

stdin **stdout** -file --opt **--help** **--version**

`pgrep [`*`параметр(ы)`*`] `*`шаблон`*

Команда `pgrep` выводит PID, соответствующие указанному *шаблону*. (Аналогичной командой является `pidof`, которая ищет только фиксированные строки.) Например, выведем идентификаторы всех запущенных процессов `python`:

```
→ pgrep python
4675
79493
82866
83114
```

Добавим параметр `-c`, чтобы подсчитать процессы:

```
→ pgrep -c python
4
```

Можно ограничить результаты процессами, запущенными определенным пользователем:

```
→ pgrep -u smith
79493
```

Команда наиболее полезна для передачи списка связанных PID другой команде. Например, вы можете найти процессы, соответствующие некой строке, с помощью `ps` и подстановки команд (см. подраздел «Подстановка команд» главы 1):

```
→ ps -fwp $(pgrep python)
```

Больше примеров использования `pgrep` с командой `kill` приведено в разделе «Управление процессами» главы 3.

Полезные параметры

`-d` *строка*	В качестве *строки* указывается символ-разделитель между PID (по умолчанию — перевод строки)
`-u` *пользователь*	Перечисление процессов, запущенных под указанным именем или идентификатором *пользователя* (UID). Для реального идентификатора пользователя примените `-U`
`-f`	Поиск по полной командной строке процесса, а не только по имени
`-x`	Точное совпадение, а не совпадение подстрок
`-v`	Вывод идентификаторов *не*совпадающих процессов

---

uptime

stdin **stdout** -file --opt **--help** **--version**

```
uptime
```

Команда `uptime` сообщает время работы системы с момента запуска:

```
→ uptime
 10:54pm up 8 days, 3:44, 3 users,
 load average: 0.89, 1.00, 2.15
```

Команда выдает полную информацию: текущее время (22:54), время работы системы (8 дней, 3 часа, 44 минуты), количество пользователей, вошедших в систему (3), средняя загрузка системы за три периода времени — 1 минуту (0,89), 5 минут (1,00) и 15 минут (2,15). Среднее значение нагрузки — это количество процессов, готовых к запуску в данный промежуток времени.

---

w

stdin **stdout** -file **--opt** **--help** **--version**

```
w [пользователь]
```

Каждый пользователь, вошедший в оболочку, может посмотреть текущие процессы с помощью команды `w`:

```
→ w
 10:51pm  up 8 days,  3:42,  8 users,
 load average: 2.02, 3.79, 5.44
USER     TTY   FROM  LOGIN@  IDLE   JCPU   PCPU  WHAT
barrett pts/0 :0    Sat 2pm 27:13m 0.07s  0.07s emacs
jones   pts/1 host1 6Sep03   2:33m 0.74s  0.21s bash
smith   pts/2 host2 6Sep03   0.00s 13.35s 0.04s w
```

Верхняя строка аналогична выводу команды `uptime`. В столбцах указывается терминал пользователя, исходный хост или X-дисплей, время входа в систему, время простоя, два показателя процессорного времени (чтобы узнать подробности, запустите команду `man w`) и текущий процесс. Для получения информации об определенном пользователе укажите имя *пользователя*.

Для получения краткого вывода задействуйте команду `w -hfs`.

Полезные параметры

`-h`	Без вывода заголовков
`-f`	Без вывода столбца FROM
`-s`	Без вывода столбцов JCPU и PCPU

---

top

stdin **stdout** -file --opt **--help** **--version**

`top [`*параметр(ы)*`]`

Команда `top` отслеживает наиболее активные процессы, обновляя информацию на экране через регулярные промежутки времени (по умолчанию каждые 3 секунды):

```
→ top
top - 13:02:14 up 1 day, 32 min, 9 users, ...
Tasks: 719 total, 2 running, 717 sleeping, 0 stopped...
%Cpu(s): 0.3 us, 0.2 sy, 0.0 ni, 99.6 id, 0.0 wa, ...
MiB Mem :  31950.5 total,  3040.6 free, 5267.4 used ...
MiB Swap:  32000.0 total, 31991.7 free,    8.2 used ...

PID   USER  PR NI VIRT SHR S  %CPU %MEM TIME CMD
26265 smith 20  0 1092 840 R   4.7  0.2 0:00 top
```

```
    1 root  20  0  540 472 S   0.0  0.1 0:07 systemd
  914 www    0  0    0   0 SW  0.0  0.0 0:00 httpd
⋮
```

Во время работы команды `top` можно динамически изменять ее поведение, нажимая разные клавиши (`s` для повышения скорости обновления, `i` для сокрытия простаивающих процессов, `k` для завершения процессов). Нажмите `h`, чтобы вывести список сочетаний клавиш, `q` — чтобы выйти из программы. Похожие команды для мониторинга ввода-вывода и пропускной способности сети называются `iotop` и `iftop`. Также обратите внимание на универсальную команду `btop`, которая отображает общую статистику по процессорам, дискам, процессам и сетевым интерфейсам.

Полезные параметры

`-n`*N*	Выполнение *N* обновлений, а затем завершение работы программы
`-d`*N*	Обновление экрана каждые *N* секунд
`-p`*N*	Вывод процесса с идентификатором *N*. Поддерживается до 20 PID
`-c`	Вывод аргументов процессов
`-b`	Передача результата на стандартный вывод. Подходит для передачи или перенаправления вывода в файл. Чтобы сохранить один шаг цикла `top` в файл, выполните команду `top -b -n1 > outfile`

free

stdin **stdout** -file --opt **--help** **--version**

```
free [параметр(ы)]
```

Команда `free` отображает общее количество используемой памяти в килобайтах:

```
→ free
       total    used    free shared buf/cache available
Mem:  523812  491944   31868      0    224812    299000
Swap: 530104       0  530104
```

Ядро Linux резервирует память для целей кэширования (столбец `buf/cache`), поэтому свободная оперативная память в выводе отображается в столбце `available` (то есть 299 000 Кбайт), а не в столбце `free` (31 868 Кбайт).

Полезные параметры

`-s` *N*	Непрерывная работа и обновление экрана каждые *N* секунд
`-m`	Отображение ресурсов памяти в мегабайтах
`-h`	Отображение значений в удобном для человека виде
`-t`	Добавление в нижней части строки итогов

Управление процессами

`kill`	Прерывание процесса (или отправка ему сигнала)
`pkill`	Прерывание процесса по имени (или отправка ему сигнала)
`timeout`	Прерывание команды, выполняющейся слишком долго
`nice`	Вызов программы с определенном приоритетом
`renice`	Изменение приоритета задания в ходе его выполнения
`nohup`	Запуск процесса, продолжающегося после выхода из системы
`flock`	Проверка того, что в определенный момент времени выполняется только один экземпляр команды

После запуска процессов можно останавливать, перезапускать, прерывать их и определять их приоритет. Я уже рассматривал некоторые из этих операций, выполняемых оболочкой, в разделе «Управление заданиями оболочки» главы 1. Давайте продолжим разговор о прерывании и определении приоритетов процессов.

---

kill

stdin **stdout** -file --opt **--help** --version

`kill [`*параметр(ы)*`] [`*идентификатор*`]`

Команда `kill` посылает сигнал процессу по его *идентификатору*. Это может привести к завершению процесса (действие по умолчанию), прерыванию, приостановке, аварийному завершению и т. д. Чтобы повлиять на процесс, вы должны быть или его владельцем, или суперпользователем. Чтобы завершить процесс `13243`, выполните команду

```
→ kill 13243
```

Если команда не работает, добавьте параметр `-KILL` или `-9`:

```
→ kill -KILL 13243
```

Теперь команда практически гарантированно сработает. Однако это не полноценное завершение работы программы — после завершения могут остаться выделенные ресурсы или проявятся другие зависимости.

Чтобы узнать идентификатор процесса, выполните команду `ps` и посмотрите вывод. Или выполните команду `pgrep`, чтобы извлечь только идентификатор:

```
→ ps -uax | grep emacs
smith 8374  ...  emacs myfile.txt      Текущий процесс emacs
smith 9051  ...  grep emacs            Ошибочный результат,
                                       следует игнорировать
→ pgrep emacs                          Более точный вывод emacs
8374
```

Теперь прервите этот процесс по идентификатору или имени, подставив команду (см. раздел «Подстановка команд» главы 1):

```
→ kill 8374                            Прерывание по идентификатору
→ kill $(pgrep emacs)                  Прерывание по имени
```

Или используйте команду `pkill`, чтобы прервать все процессы для указанной программы:

```
→ pkill emacs
```

Кроме команды `kill` в файловой системе (обычно `/bin/kill`), большинство оболочек имеют встроенные команды `kill`. Их синтаксис и поведение различаются, но все они поддерживают следующий синтаксис:

```
kill -N PID
kill -ИМЯ PID
```

где *`N`* — номер сигнала, *`ИМЯ`* — имя сигнала без ведущего `SIG` (например, чтобы послать сигнал `SIGHUP`, используйте `-HUP`), а *`PID`* — идентификатор процесса, который нужно прервать. Чтобы увидеть полный список сигналов, передаваемых `kill`, выполните команду `kill -l`, хотя у вас вывод может отличаться от приведенного здесь в зависимости от того, какой именно `kill` вы запускаете. Для просмотра описания сигналов выполните `man 7 signal`.

timeout

stdin stdout -file --opt **--help** **--version**

```
timeout [параметр(ы)] время команда
```

Команда `timeout` устанавливает ограничение на *время* выполнения программы в секундах. Если программа выполняется дольше установленного лимита, то команда прервет выполнение. В качестве примера приведена команда `sleep`, которая выполняется в течение 1 минуты, но будет прервана через 3 секунды:

```
→ timeout 3 sleep 60          Завершение через 3 секунды
```

В качестве практического примера можно привести воспроизведение MP3-музыки из вашей коллекции в течение часа (3600 секунд), а затем прерывание процесса:

```
→ timeout 3600 mplayer *.mp3
```

Полезные параметры

`-s` *сигнал* — Отправка *сигнала*, отличного от сигнала по умолчанию (TERM). Варианты выбора те же, что и в `kill -l`

`-k` *время* — Если выполнение программы не прерывается после первого сигнала, через указанный промежуток *времени* в секундах передается сигнал `KILL`

nice

stdin **stdout** -file --opt **--help** **--version**

```
nice [-n приоритет] программа
```

При вызове требовательной к системе программы будьте осторожнее с остальными процессами (и пользователями). Понизьте приоритет этой *программы* с помощью команды `nice` — она устанавливает уровень *приоритета* для процесса, чтобы планировщик процессов Linux обращал на него меньше внимания. Далее представлен пример установки большого задания на уровень приоритета 7:

```
→ nice -n 7 sort hugefile > outfile
```

Обычные процессы (запущенные без `nice`) выполняются на уровне приоритета 0. Вы можете убедиться в этом, если выполните команду `nice` без аргументов:

```
→ nice
0
```

Если вы опустите параметр `-n`, уровень приоритета по умолчанию будет равен `10`. Кроме того, суперпользователь может понизить уровень приоритета процесса `nice`, тем самым повысив приоритет программы:

```
→ sudo nice -n -10 программа
```

Уровни приоритета процессов указываются в выводе `ps` (столбец `NI`) и `top` (столбец `N`):

```
→ ps -o pid,user,args,nice
```

Если программа часто обращается к диску, стоит обратить внимание на команду `ionice`, аналогичную программе `nice` для ввода/вывода.

---

renice

stdin **stdout** -file --opt **--help** **--version**

```
renice [-n N] [параметр(ы)] идентификатор
```

В то время как команда `nice` вызывает программу с указанным уровнем приоритета, `renice` изменяет уровень приоритета уже запущенного процесса. В качестве быстрого теста создайте процесс, который бездействует в течение 2 минут. Затем запустите процесс в фоновом режиме и поднимите уровень его приоритета до 5:

```
→ sleep 120 &
→ pgrep sleep
2673
→ renice -n 5 -p 2673
2673 (process ID) old priority 0, new priority 5
```

Обычные пользователи могут повышать уровень приоритета для своих процессов, а суперпользователь — понижать его и управлять любым процессом. Диапазон допустимых

значений от –20 до +20, но избегайте больших отрицательных значений, иначе вы можете нарушить работу важных системных процессов.

Полезные параметры

`-p` *идентификатор*	Изменение приоритета процесса с указанным *идентификатором*. Вы можете опустить параметр `-p` и просто указать идентификатор (`renice -n 5 28734`)
`-u` *пользователь*	Прерывание всех процессов, принадлежащих указанному *пользователю*

nohup

stdin stdout -file --opt --help --version

`nohup` *команда*

Используйте команду `nohup`, чтобы продолжить выполнение *команды* после завершения оболочки. Обычно, когда оболочка или другой процесс завершаются, дочерние процессы посылают сигнал завершения. Команда `nohup`, которая расшифровывается как *no hangup*, позволяет команде игнорировать сигналы завершения:

```
→ nohup долго_выполняющаяся_команда &
```

Если команда выводит результат в `stdout` или `stderr`, `nohup` переводит ее вывод в файл `nohup.out`, находящийся в текущем каталоге (если у вас достаточно прав) или в вашем домашнем каталоге (если их нет).

flock

stdin stdout -file --opt --help --version

`flock [`*параметр(ы)*`] файл_блокировки команда`

Иногда бывают ситуации, когда нужно убедиться, что на компьютере одновременно работает только один экземпляр программы. Например, если вы запускаете автоматическое резервное копирование каждый час с помощью команды типа `rsync`, то есть небольшая вероятность дублирования процесса (предыдущая резервная копия все еще создается при запуске

следующей). Команда `flock` решает подобную проблему. Она создает *файл_блокировки*, который не позволяет *команде*, например сценарию резервного копирования, выполняться повторно одновременно с самой собой. Если попытаться запустить сразу две копии команды с одним и тем же файлом блокировки, вторая команда не будет выполнена. Например, команда `rsync`, запущенная с помощью `flock` и файла блокировки `/tmp/mylock`, мгновенно неудачно завершается, если уже запущен другой экземпляр этой же команды:

```
→ flock -n /tmp/mylock rsync -av dir1 dir2
```

Чтобы увидеть команду `flock` в действии, откройте два окна оболочки и выполните в каждом из них по очереди следующую команду (в качестве примера я использую команду `sleep`, которая не делает ничего — просто задает режим ожидания в течение указанного количества секунд):

```
→ flock -n /tmp/mylock sleep 60
```

Первая команда выполняется, а вторая мгновенно завершается, так как у них один и тот же *файл блокировки*. Это может быть любое имя файла или каталога, которое `flock` рассматривает как уникальный маркер, предотвращающий выполнение других команд. Например, если вы запустили предыдущую команду `sleep` в одной оболочке и другую команду, например `ls`, в другой оболочке с тем же файлом блокировки:

```
→ flock -n /tmp/mylock ls
```

то `flock` предотвратит выполнение второй команды (`ls`).

Полезные параметры

`-n` Немедленное неудачное завершение, если уже запущена другая команда

`-w N` Неудачное завершение через *N* секунд ожидания, если уже запущена другая команда

`-s` Общая блокировка. С помощью этого параметра можно выполнять несколько команд одновременно, но если не использовать параметр, то `flock` будет неудачно завершаться. Это полезно для ограничения количества команд, которые могут выполняться одновременно

Планирование заданий

`sleep`	Бездействие в течение указанного количества секунд
`watch`	Запуск команды через указанные промежутки времени
`at`	Планирование однократного выполнения задания на будущее
`crontab`	Планирование многократного выполнения задания на будущее

Если вам нужно запускать программы в определенное время или через регулярные промежутки времени, Linux предоставляет несколько инструментов планирования.

sleep

stdin stdout -file --opt **--help** **--version**

```
sleep время
```

Команда `sleep` выжидает указанное *время*. Оно может быть представлено целым числом (то есть секундами) или целым числом с буквой `s` (секунды), `m` (минуты), `h` (часы) или `d` (дни), например:

```
→ sleep 5m                    Бездействие на протяжении 5 минут
```

Спящий режим полезен для отсрочки выполнения команды на определенное время:

```
→ sleep 3 && echo 'Three seconds have passed.'  (проходит 3 секунды)
Three seconds have passed
```

watch

stdin **stdout** -file **--opt** **--help** **--version**

```
watch [параметр(ы)] команда
```

Программа `watch` выполняет *команду* через регулярные промежутки времени, по умолчанию — через каждые 2 секунды. Команда передается в оболочку (так что не забудьте заключить в кавычки или экранировать все специальные символы), а результаты отображаются в полноэкранном режиме. Например, `watch -n 1 date` выполняет команду `date` раз в секунду — что-то вроде электронных часов забывчивого. Нажмите сочетание клавиш `^C` для выхода.

Полезные параметры

`-n` *время*	Промежуток между выполнениями в секундах
`-d`	Подсветка различий в выводе, чтобы подчеркнуть, что изменилось от одного выполнения к другому
`-g`	Завершение работы, когда команда выдает результат, отличающийся от предыдущего выполнения

---

at

stdin **stdout** -file **--opt** --help --version

`at [`*параметр(ы)*`] `*дата/время*

Команда `at` планирует выполнение одной или нескольких команд интерпретатора для последующего запуска:

```
→ at 7am next sunday
at> echo Remember to go shopping | mail smith
at> lpr $HOME/shopping-list
at> ^D
<EOT>
job 559 at 2024-09-08 21:30
```

Если в запланированное время хост выключен или находится в спящем режиме, задание будет запущено сразу после его запуска. Временные спецификации, поддерживаемые командой `at`, очень гибкие:

- время, за которым следует дата (а не дата, за которой следует время);
- только дата (текущее время на часах);
- только время (следующее событие, которое произойдет сегодня или завтра);
- ключевое слово, например `now`, `midnight` или `teatime` (16:00);
- любое из предыдущих, за которым следует сдвиг, например «+3 дня».

Даты могут принимать различные формы: `december 25 2030`, `25 december 2030`, `december 25`, `25 december`, `12/25/2030`, `25.12.2030`, `20301225`, `today`, `thursday`, `next thursday`, `next month`, `next year` и т. д. Названия месяцев могут быть

сокращены до трех букв (`jan`, `feb`, `mar` и т. д.). Время также указывается в любом формате: `8pm`, `8 pm`, `8:00pm`, `8:00 pm`, `20:00` или `2000`. Сдвиг — это знак «плюс» или «минус», за которым следует пробел и временной промежуток: `+ 3 days`, `+ 2 weeks`, `– 1 hour` и т. д.[1]

Если вы опускаете часть даты или времени, `at` скопирует недостающую информацию из системных часов. Таким образом, `next year` означает следующий год, `thursday` — ближайший четверг по текущему времени, `december 25` — ближайшее 25 декабря, а `4:30pm` — ближайшее время 16:30.

Команда, вводимая в `at`, не оценивается оболочкой до момента выполнения, так что шаблоны файлов, переменные и другие конструкции оболочки не расширяются заранее. Кроме того, ваше текущее окружение (см. `printenv`) сохраняется в каждом задании и будет выполняться так, как будто вы вошли в систему. Однако для заданий `at` нельзя использовать псевдонимы.

Чтобы вывести список заданий `at`, выполните команду `atg` (at queue):

```
→ atq
559  2024-09-08 21:30 a smith
```

Чтобы удалить задание, выполните команду `atrm` (at remove) с указанием номера задания:

```
→ atrm 559
```

Полезные параметры

`-f` *файл*	Выполнение команды из указанного *файла*, а не из стандартного ввода
`-c` *номер*	Вывести команды задания с указанным *номером* на стандартный вывод

[1] См. синтаксис в файле /usr/share/doc/at/timespec.

crontab

stdin **stdout** **-file** **--opt** --help --version

```
crontab [параметр(ы)] [файл]
```

Команда `crontab`, как и команда `at`, планирует задания на определенное время. Однако `crontab` предназначена для повторяющихся заданий, например «Выполнять команду в полночь каждый вторник». Чтобы команда работала, вам нужно отредактировать и сохранить файл `crontab`, который автоматически устанавливается в системный каталог (`/var/spool/cron`). Раз в минуту процесс Linux `cron` просыпается, проверяет все файлы `crontab` и выполняет все запланированные задания.

`crontab -e`

Редактирование файла `crontab` в редакторе по умолчанию (`$VISUAL`).

`crontab -l`

Передача содержимого файла `crontab` на стандартный вывод.

`crontab -r`

Безвозвратное удаление файла `crontab`.

`crontab myfile`

Назначение файла `myfile` программе `crontab`.

Суперпользователь может добавить параметр `-u` *пользователь*, чтобы работать с файлами `crontab` других *пользователей*.

Файлы `crontab` содержат по одному заданию в строке. (Пустые строки и строки, начинающиеся с символа `#`, игнорируются.) Каждая строка содержит шесть столбцов, разделенных пробелами. Первые пять столбцов — это время выполнения задания, а последний — сама команда задания.

Минуты часа

Целые числа от 0 до 59. Это может быть одно число (`30`), последовательность чисел, разделенных запятыми

(`0,15,30,45`), диапазон чисел (`20-30`), последовательность диапазонов (`0-15,50-59`) или звездочка, означающая «все». Вы также можете указать «каждый *n*-й раз» с помощью суффикса `/n`. Например, `*/12` и `0-59/12` означают `0,12,24,36,48`, то есть каждые 12 минут.

Часы дня

Синтаксис тот же, что и для минут.

Дни месяца

Целые числа от 1 до 31. Вы можете использовать последовательности, диапазоны, последовательности диапазонов или звездочку.

Месяцы года

Целые числа от 1 до 12. Можно применять последовательности, диапазоны, последовательности диапазонов или звездочку. Кроме того, вы можете использовать трехбуквенные сокращения (`jan`, `feb`, `mar` и т. д.), но только не в диапазонах или последовательностях.

Дни недели

Целые числа от 0 (воскресенье) до 6 (суббота). Можно использовать последовательности, диапазоны, последовательности диапазонов или звездочку. Кроме того, вы можете применять трехбуквенные сокращения (`sun`, `mon`, `tue` и т. д.), но только не в диапазонах или последовательностях.

Команда для выполнения

Любая команда оболочки. Она выполняется при входе в систему, поэтому вы можете включить переменные окружения, например `$HOME`, и они будут работать. Используйте только абсолютные пути к вашим командам (например, `/usr/bin/who` вместо `who`), чтобы `cron` точно запустила нужные программы, так как в системе Linux может быть несколько программ с одинаковым именем.

Вот некоторые примеры спецификаций времени:

`*`	`*`	`*`	`*`	`*`	*Каждую минуту*
`45`	`*`	`*`	`*`	`*`	*45-ю минуту каждого часа (то есть 1:45, 2:45 и т. д.)*
`45`	`9`	`*`	`*`	`*`	*Каждый день в 9:45*
`45`	`9`	`8`	`*`	`*`	*Восьмой день каждого месяца в 9:45*
`45`	`9`	`8`	`12`	`*`	*Каждое 8 декабря в 9:45*
`45`	`9`	`8`	`dec`	`*`	*Каждое 8 декабря в 9:45*
`45`	`9`	`*`	`*`	`6`	*Каждую субботу в 9:45*
`45`	`9`	`*`	`*`	`sat`	*Каждую субботу в 9:45*
`45`	`9`	`*`	`12`	`6`	*Каждую субботу декабря в 9:45*
`45`	`9`	`8`	`12`	`6`	*Каждую субботу декабря, а также 8 декабря, в 9:45*

Далее показана полная запись для запуска сценария каждую субботу в 9:45:

```
45 9 * * sat /usr/local/bin/myscript
```

Если задание печатает какой-либо вывод, `cron` отправит копию владельцу файла `crontab`.

СОВЕТ

Избегайте использования длинных команд оболочки в файле crontab. Вместо этого храните команды в сценариях оболочки для запуска из crontab.

Вход, выход и выключение системы

`systemctl`	Контроль состояния машины и ее служб
`shutdown`	Выключение локальной машины
`reboot`	Перезагрузка локальной машины

Вход и выход из GNOME, KDE или других графических Рабочих столов очень прост. Чтобы выйти из оболочки, просто закройте ее (выполните команду `exit` или `logout` или вручную нажмите сочетание клавиш `^D` в строке). В Linux также есть команды для перезагрузки и выключения компьютера

или отдельных служб. Никогда не выключайте питание Linux-системы просто так — для сохранения файловой системы требуется завершать работу специальным способом.

systemctl

stdin **stdout** -file --opt **--help** **--version**

```
systemctl [параметр(ы)] команда [аргумент(ы)]
```

Команда `systemctl` управляет системными службами. Она является частью менеджера служб `systemd`. Полное описание `systemd` выходит за рамки тематики этой книги, так что я расскажу только об основных вариантах ее использования. (Дополнительные сведения ищите в `man systemd`.)

`systemctl` может управлять системой в целом.

`sudo systemctl poweroff`	Выключение системы
`sudo systemctl reboot`	Перегрузка системы
`sudo systemctl suspend`	Приостановка работы системы

Также она может управлять отдельными службами, например веб-серверами и базами данных.

`systemctl`	Перечисление *служб* и их состояний
`sudo systemctl enable` *служба*	Подготовка *службы*, без запуска
`sudo systemctl start` *служба*	Запуск включенной *службы*
`sudo systemctl restart` *служба*	То же, что и *stop*, за которой следует *start*
`sudo systemctl reload` *служба*	Перезагрузка конфигурации работающей *службы*
`sudo systemctl status` *служба*	Вывод состояния *службы*. Для получения подробной информации о состоянии обратитесь к *man journalctl*
`sudo systemctl stop` *служба*	Остановка запущенной службы
`sudo systemctl disable` *служба*	Отключение службы

Имена служб имеют суффикс `.service`, который можно опустить. Например, чтобы перезапустить сервер базы данных mySQL, подойдет любая из следующих команд:

```
→ sudo systemctl restart mysqld.service      С суффиксом
→ sudo systemctl restart mysqld              Без суффикса
```

shutdown

stdin **stdout** -file --opt **--help** --version

```
shutdown [параметр(ы)] время [сообщение]
```

Команда `shutdown` останавливает или перезагружает машину Linux, выполнять ее может только суперпользователь. (Во многих дистрибутивах Linux команда `shutdown` является символической ссылкой на `systemctl`.) Приведу пример: команда должна остановить систему через 10 минут, передав *сообщение* «плановое обслуживание» всем вошедшим в систему пользователям:

```
→ sudo shutdown -h +10 "плановое обслуживание"
```

Временем может быть число минут со знаком «плюс», например `+10`. Это может быть абсолютное время в часах и минутах, например `16:25`, или слово `now`, означающее «немедленно».

Без параметров команда `shutdown` переводит систему в однопользовательский режим — специальный режим обслуживания, когда только суперпользователь может войти в систему, а все несущественные службы отключены. Чтобы выйти из этого режима, нужно либо снова выполнить команду `shutdown` для остановки или перезагрузки системы, либо выйти из оболочки с помощью `exit` или `^D` для загрузки системы в обычном многопользовательском режиме.

Полезные параметры

`-r` Перезапуск системы

`-h` Остановка работы системы

`-k` Шутка — на самом деле команда не выключает систему, а просто передает всем пользователям предупреждающее сообщение о том, что система якобы выходит из строя

`-c` Отмена выполняющегося выключения (игнорирование аргумента *время*)

`-f` При перезагрузке пропуск обычной проверки файловой системы, выполняемой командой `fsck` (см. раздел «Использование дисков и файловых систем» главы 4)

`-F` При перезагрузке выполнение обычной проверки файловой системы

Для получения технической информации о выключении, однопользовательском режиме и различных состояниях системы обратитесь к `man systemd` или `man init`.

reboot

stdin **stdout** -file --opt **--help** --version

```
reboot [параметр(ы)]
```

Команда `reboot` немедленно перезагружает компьютер. Она требует прав суперпользователя, но в некоторых дистрибутивах ее может выполнить любой пользователь.

У `reboot` есть параметры (см. документацию), но я их практически не применяю. В некоторых дистрибутивах Linux команда `reboot` — это просто символическая ссылка на `systemctl`.

Пользователи и их окружение

`logname`	Вывод имени пользователя
`whoami`	Вывод текущего имени пользователя
`id`	Вывод идентификатора пользователя и его принадлежности группе
`who`	Вывод списка вошедших в систему пользователей, подробно
`users`	Вывод списка вошедших в систему пользователей, кратко
`tty`	Вывод имени терминального устройства
`last`	Определение последнего авторизованного в системе пользователя
`printenv`	Вывод окружения

Кто вы? Только система знает ответ на этот вопрос. Набор специальных программ сообщит вам все о ее *пользователях*: имена, время входа в систему и окружение.

logname

stdin **stdout** -file --opt **--help** --version

```
logname
```

Команда `longname` выводит ваше имя для входа в систему:

```
→ logname
smith
```

Если эта команда не работает, попробуйте выполнить вот эту:

```
→ echo $LOGNAME
```

whoami

stdin **stdout** -file --opt **--help** **--version**

stdin **stdout** - -- **--help** **--version**
whoami

Команда `whoami` выводит имя текущего пользователя. Если вы задействуете команду `sudo`, то ваше имя может отличаться от имени для входа в систему (вывод `logname`). Вот чем `whoami` отличается от `logname`:

```
→ logname                          Имя пользователя
smith
→ sudo logname                     Имя не изменяется
smith
→ whoami                           Текущее имя пользователя
smith
→ sudo whoami                      Имя меняется
root
```

id

stdin **stdout** -file **--opt** **--help** **--version**

`id [`*`параметр(ы)`*`] [`*`пользователь`*`]`

Каждый *пользователь* имеет уникальный числовой *пользовательский идентификатор*, а группа — уникальный числовой *идентификатор группы*. Команда `id` выводит эти значения вместе с соответствующими именами пользователей и групп:

```
→ id
uid=500(smith) gid=500(smith) groups=500(smith),6(disk)
```

Полезные параметры

`-u` Вывод текущего идентификатора пользователя и завершение работы

`-g` Вывод идентификатора группы и завершение работы

`-G` Вывод идентификаторов всех групп, к которым принадлежит пользователь

К любому из предыдущих параметров добавьте `-n`, чтобы вывести имена пользователей и групп вместо идентификаторов, или `-r`, чтобы вывести реальные идентификаторы/имена, а не текущие.

---

who

stdin **stdout** -file --opt **--help** **--version**

`who [`*`параметр(ы)`*`] [`*`файл`*`]`

Команда `who` выводит список всех пользователей, вошедших в систему. Пользователи с несколькими интерактивными оболочками отображаются несколько раз:

```
→ who
Smith    pts/0   Sep  6 17:09 (:0)
barrett  pts/1   Sep  6 17:10 (10.24.19.240)
jones    pts/2   Sep  8 20:58 (192.168.13.7)
jones    pts/4   Sep  3 05:11 (192.168.13.7)
```

Обычно `who` считывает данные из файла `/var/run/utmp`. Вы можете указать другой *`файл`*, например `/var/log/wtmp` для успешно авторизованных логинов или `/var/log/btmp` для неудачных аутентификаций[1].

Полезные параметры

`-H`	Вывод заголовков в первой строке
`--lookup`	Перечисление имен хостов удаленных пользователей, вошедших в систему
`-u`	Вывод времени простоя терминала каждого пользователя
`-m`	Вывод информации только о пользователе активного терминала
`-q`	Краткий вывод количества и имен пользователей. По аналогии с командой `users`, но со счетчиком

[1] Если ваша система настроена на регистрацию этой информации.

users

stdin **stdout** -file --opt **--help** **--version**

```
users [файл]
```

Команда `users` выводит краткий список пользователей, вошедших в систему. Пользователи с несколькими интерактивными оболочками отображаются несколько раз:

```
→ users
barrett jones smith smith smith
```

Как и команда `who`, `users` по умолчанию считывает файл `/var/log/utmp`, но может считывать данные и из другого предоставленного *файла*.

tty

stdin **stdout** -file --opt **--help** **--version**

```
tty
```

Команда `tty` выводит имя терминального устройства, связанного с текущей оболочкой:

```
→ tty
/dev/pts/4
```

last

stdin **stdout** -file **--opt** **--help** **--version**

```
last [параметр(ы)] [пользователь(и)] [терминал(ы)]
```

Команда `last` отображает историю входов в систему в обратном хронологическом порядке:

```
→ last
bob pts/3 localhost Mon Sep 8 21:07 - 21:08 (00:01)
sue pts/6 :0        Mon Sep 8 20:25 - 20:56 (00:31)
bob pts/4 myhost    Sun Sep 7 22:19 still logged in
⋮
```

Вы можете указать имена *пользователей* или имя *терминального* устройства, чтобы отфильтровать вывод.

Полезные параметры

`-`*N*	Вывод последних *N* строк вывода, где *N* — целое положительное число
`-p` *время*	Вывод пользователей, вошедших в систему в указанное *время*. Для получения информации о текущих входах выполните команду `last -p now`
`-i`	Вывод IP-адресов вместо имен хостов
`-R`	Сокрытие имен хостов
`-x`	Вывод информации о завершениях работы системы и изменениях уровня ее запуска (например, из однопользовательского режима в многопользовательский)
`-f` *файл*	Чтение из *файла* данных, отличного от `/var/run/wtmp`. Более подробную информацию можно получить с помощью команды `who`

---

printenv

stdin **stdout** -file **--opt** **--help** **--version**

```
printenv [переменная]
```

Команда `printenv` выводит на печать все *переменные* окружения, известные вашей оболочке, и их значения:

```
→ printenv
HOME=/home/smith
MAIL=/var/spool/mail/smith
NAME=Sandy Smith
SHELL=/bin/bash
⋮
```

или только указанные *переменные*:

```
→ printenv HOME SHELL
/home/smith
/bin/bash
```

Управление учетными записями пользователей

`useradd`	Создание учетной записи
`userdel`	Удаление учетной записи
`usermod`	Изменение учетной записи
`passwd`	Добавление пароля
`chsh`	Изменение оболочки пользователя

Программа установки Linux автоматически создает учетную запись суперпользователя и обычную учетную запись пользователя. Вы можете создать и другие учетные записи. Просто помните, что каждая учетная запись — это потенциальная возможность для злоумышленника проникнуть в вашу систему. При создании учетных записей не забудьте придумать надежные пароли.

useradd

stdin **stdout** -file **--opt** **--help** --version

`useradd [`*`параметр(ы)`*`] `*`пользователь`*

Команда `useradd` позволяет суперпользователю создать учетную запись пользователя (не путайте ее с аналогичной командой `adduser`):

```
→ sudo useradd -m smith
```

Значения по умолчанию вряд ли пригодятся (чтобы увидеть их, запустите команду `useradd -D`), поэтому не забудьте указать все желаемые параметры, например:

```
→ sudo useradd -d /home/smith -s /bin/bash -G games,video \
  smith
```

Полезные параметры

`-m`	Создание домашнего каталога пользователя и копирование в него нескольких стандартных файлов из каталога шаблонов `/etc/skel`. Каталог шаблонов содержит базовые версии файлов инициализации, например `~/.bashrc`, для начала работы новых пользователей. Если вы хотите скопировать файлы из другого каталога, добавьте параметр `-k` (`-k` *каталог*)
`-d` *каталог*	Выбор домашнего *каталога* пользователя
`-s` *оболочка*	Выбор *оболочки* входа пользователя в систему
`-u` *идентификатор*	Определение *идентификатора* пользователя
`-c` *строка*	Определение поля комментария пользователя (называется *полем GECOS*). Обычно это полное имя пользователя, но может быть и любая *строка*
`-g` *группа*	Определение начальной *группы* пользователя (по умолчанию), которая указывается числовым идентификатором или именем уже существующей группы
`-G` *группа1*`,`*группа2*…	Добавление пользователя в существующую *группу1*, *группу2* и т. д.

userdel

stdin **stdout** -file **--opt** **--help** --version

```
userdel [-r] пользователь
```

Команда `userdel` удаляет существующего пользователя:

```
→ sudo userdel smith
```

Команда не удаляет файлы пользователя (домашний каталог, почтовый ящик и т. д.), если не указан параметр `-r`. Хорошо подумайте перед тем, как удалять пользователя, — лучше отключить учетную запись с помощью команды `usermod -L`. Убедитесь, что у вас есть резервные копии всех файлов пользователя, — возможно, когда-нибудь они вам понадобятся.

usermod

stdin **stdout** -file **--opt** **--help** --version

```
usermod [параметр(ы)] пользователь
```

Команда `usermod` настраивает учетную запись определенного пользователя разными способами, например изменяет домашний каталог:

```
→ sudo usermod -d /home/another smith
```

Полезные параметры

`-a`	При добавлении пользователя в группу (`-G`) сохраняется его существующее членство в группах
`-c` *строка*	Изменение поля комментария пользователя (поля GECOS). Обычно это полное имя пользователя, но может быть и любая *строка*
`-d` *каталог*	Изменение домашнего каталога пользователя на указанный
`-l` *пользователь*	Изменение имени пользователя на указанное. **Внимание:** от этого пользователя ничего не должно зависеть в системе! Также не меняйте системные учетные записи (root, daemon и т. д.), если не понимаете, что делаете!
`-s` *оболочка*	Изменение оболочки входа пользователя в систему на указанную

`-g` *группа*	Изменение начальной группы пользователя (по умолчанию) на указанную. Может быть представлена числовым идентификатором или именем уже существующей группы
`-G` *группа1*`,`*группа2*...	Внесение пользователя в ряды членов *уже* существующей *группы1*, *группы2* и т. д. **Внимание:** если пользователь в настоящее время состоит в других группах, но вы их не указали, то команда `usermod` исключит его из других групп. Чтобы предотвратить такое поведение и сохранить существующие группы пользователя, добавьте параметр `-a`
`-L`	Отключение учетной записи, чтобы пользователь не мог войти в систему
`-U`	Активация учетной записи после ее отключения (`-L`)

passwd

stdin **stdout** -file **--opt** **--help** --version

```
passwd [параметр(ы)] [пользователь]
```

Команда `passwd` изменяет пароль для входа в систему, который по умолчанию является вашим паролем:

```
→ passwd
```

или пароль другого *пользователя*, если команду запускает суперпользователь:

```
→ sudo passwd smith
```

Команда `passwd` поддерживает параметры, большинство которых связано с истечением срока действия пароля. Применяйте их только при крайней необходимости.

chsh

stdin **stdout** -file **--opt** **--help** **--version**

```
chsh [параметр(ы)] [пользователь]
```

Команда `chsh` (*change shell*) устанавливает вашу оболочку для входа в систему. Если команда вызывается без указания имени *пользователя*, то она влияет на вашу учетную запись, а если с указанием имени *пользователя* (от имени суперпользователя), то влияет на указанного пользователя.

При отсутствии параметров `chsh` запрашивает нужную информацию:

```
→ chsh
Password: xxxxxxxx
New shell [/bin/bash]: /bin/tcsh
```

Новая оболочка должна быть указана в файле `/etc/shells`.

Полезные параметры

`-s` *оболочка*	Установка указанной *оболочки*
`-l`	Вывод всех доступных оболочек, установленных в системе

Управление группами

`groups`	Вывод членства пользователя в группах
`groupadd`	Создание группы
`newgrp`	Незамедлительная активация членства в группе
`groupdel`	Удаление группы
`groupmod`	Изменение группы

Группа — это набор учетных записей, рассматриваемых как единое целое. Если вы предоставляете группе права на выполнение действия, например изменение файла, то все ее члены могут это сделать. Например, предоставим группе «Друзья» полные права на чтение, изменение и выполнение файла `/tmp/sample`:

```
→ groups
users smith friends
→ chgrp friends /tmp/sample
→ chmod 770 /tmp/sample
→ ls -l /tmp/sample
-rwxrwx--- 1 smith friends 2874 ... /tmp/sample
```

Чтобы добавить пользователей в группу, выполните команду `usermod -aG` или отредактируйте файл `/etc/group` от имени суперпользователя. Чтобы изменить групповую политику владения файлом, обратитесь к команде `chgrp` из раздела «Свойства файлов» главы 2.

groups

stdin **stdout** -file --opt **--help** **--version**

```
groups [пользователи]
```

Команда `groups` выводит группы Linux, к которым принадлежите вы или указанные *пользователи*:

```
→ whoami
smith
→ groups
smith users
→ groups jones root
jones : jones users
root : root bin daemon sys adm disk wheel src
```

groupadd

stdin stdout -file **--opt** **--help** --version

```
groupadd [параметр(ы)] группа
```

Команда `groupadd` создает группу. (Не путайте с аналогичной командой `addgroup`.) В большинстве случаев следует добавить параметр `-f`, чтобы предотвратить создание дубликатов групп:

```
→ sudo groupadd -f friends
```

Полезные параметры

`-g` *идентификатор*	Позволяет указать числовой *идентификатор* группы. Обычно определяется командой `groupadd`
`-f`	Если указанная группа уже существует, вывод соответствующего сообщения и завершение работы

newgrp

stdin stdout -file --opt **--help** --version

```
newgrp [-] [группа]
```

Если вас добавляют в новую группу (например, с помощью `usermod -aG`), то изменения не вступают в силу до вашего следующего входа в систему. Команда `newgrp` позволяет избежать этой проблемы. Запустите команду `newgrp` с именем

новой *группы* в качестве аргумента, и она обновит оболочку с идентификатором вашей текущей группы. Таким образом вы сможете сразу же использовать вновь предоставленные привилегии группы. Эффект сохраняется на время работы новой оболочки, но это лучше, чем выходить из системы/заходить в нее. Чтобы восстановить идентификатор группы по умолчанию, выйдите из оболочки.

Предположим, вас только что добавили в группу `video`. Если вы еще не вышли из системы, то не увидите группу `video` среди своих групп:

```
→ groups                          Просмотр активных групп
smith sudo docker
```

Запустите `newgrp`, чтобы установить идентификатор группы на `video`:

```
→ newgrp video                    Запуск оболочки с идентификатором группы video
→ groups                          Теперь video — группа по умолчанию
video sudo docker smith
→ exit                            Установка идентификатора группы по умолчанию
```

Единственный параметр — дефис — позволяет команде `newgrp` настроить окружение так, словно вы только что вошли в систему (аналогично `su -l`).

groupdel

stdin stdout -file **--opt** **--help** --version

```
groupdel группа
```

Команда `groupdel` удаляет существующую *группу*:

```
→ sudo groupdel friends
```

Перед удалением группы идентифицируйте все принадлежащие ей файлы, чтобы их можно было удалить:

```
→ sudo find / -group friends -print > /tmp/friend.files
```

Дело в том, что `groupdel` не изменяет права группы на какие-либо файлы. Она просто удаляет имя группы из `/etc/group`. Все файлы, принадлежащие удаленной группе, сохраняют ее идентификатор.

groupmod

stdin **stdout** -file **--opt** **--help** --version

`groupmod [параметр(ы)] группа`

Команда `groupmod` изменяет указанную *группу*, настраивая ее имя или идентификатор:

```
→ sudo groupmod -n newname friends
```

Команда не влияет на файлы, принадлежащие этой группе, — она лишь изменяет ее идентификатор или имя в записях системы.

Полезные параметры

`-n` *имя*	Изменение имени группы на указанное (безопасно)
`-g` *идентификатор*	Изменение идентификатора группы на указанный (опасно). Все файлы с оригинальным идентификатором группы будут иметь недействительные права группы, их нужно удалить

Установка ПО

`dnf`	Стандартный менеджер пакетов для файлов RPM (CentOS, Fedora, Red Hat, Rocky и т. д.)
`yum`	Старый менеджер пакетов для файлов RPM
`rpm`	Локальное управление пакетами RPM
`apt`	Стандартный менеджер пакетов для файлов DEB (Debian, Deepin, elementary OS, Kodachi, Linux Lite, MX, Mint, Nitrux, POP!_OS, Rescatux, Ubuntu, Zorin OS и т. д.)
`aptitude`	Альтернативный менеджер пакетов для файлов DEB
`dpkg`	Локальное управление DEB-пакетами
`emerge`	Портативный менеджер пакетов для Gentoo Linux
`pacman`	Менеджер пакетов для Arch Linux (а также Garuda, EndeavourOS, Manjaro и т. д.)
`zypper`	Менеджер пакетов для openSUSE
`flatpak`, `snap`	Менеджер пакетов на основе контейнеров

Ваш дистрибутив Linux поставляется с *менеджером пакетов* для установки программных пакетов через командную строку или через графический интерфейс. Звучит запутанно,

но каждый менеджер пакетов поддерживает уникальные команды и может использовать различные форматы файлов пакетов. Ваша первая задача как пользователя Linux — узнать, какой менеджер пакетов применяется в вашем дистрибутиве. Если вы не в курсе, то поможет следующая команда:

```
→ cat /etc/issue
Ubuntu 22.04.2 LTS \n \l                    Запуск Ubuntu Linux
→ more /etc/*-release
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=22.04
DISTRIB_CODENAME=jammy
⋮
```

Затем найдите в интернете менеджер пакетов вашего дистрибутива или просто выполните каждую команду управления пакетами из моего списка. Наиболее распространенными типами пакетов и менеджерами пакетов являются следующие.

Пакеты Debian, dpkg или .deb

Используются в Debian, Ubuntu и других дистрибутивах. Я расскажу о программах управления пакетами `apt`, `aptitude` и `dpkg`.

Пакеты RPM

Применяются в Red Hat, Fedora, CentOS и других дистрибутивах. Я расскажу о командах `dnf`, `yum`, `rpm` и `zypper`.

Эти и другие типы пакетов устанавливают программы в системные каталоги, например `/usr/bin`. Другой вид менеджеров пакетов использует иной способ и запускает программы в ограниченной мини-среде, называемой контейнером. Примерами служат Snap и Flatpak.

dnf

stdin **stdout** -file --opt **--help** **--version**

```
dnf [параметр(ы)] [пакет(ы)]
```

`dnf` — новый менеджер пакетов для пакетов RPM (файлы `.rpm`). В следующей таблице перечислены общие операции.

Операция	Команда dnf
Поиск необходимого пакета (поддерживаются подстановочные знаки * и ?)	`dnf search` *запрос*
Проверка того, установлен ли пакет	`dnf list installed` *пакет*
Загрузка пакета без установки	`dnf download` *пакет*
Загрузка пакета с установкой	`sudo dnf install` *пакет*
Установка файла пакета	`sudo dnf install` *файл*`.rpm`
Просмотр информации о пакете	`dnf info` *пакет*
Вывод содержимого пакета	`rpm -ql` *пакет*
Просмотр того, к какому пакету относится установленный файл	`dnf provides` */путь/к/файлу*
Обновление установленного пакета	`sudo dnf upgrade` *пакет*
Удаление установленного пакета	`sudo dnf remove` *пакет*
Перечисление всех пакетов, установленных в системе (совет: запускать в конвейере с командой `less`)	`dnf list installed`
Проверка обновлений для установленных пакетов	`dnf check-update`
Обновление всех пакетов в системе	`sudo dnf upgrade`
Обновление дистрибутива для следующей версии	`sudo dnf system-upgrade`

yum

stdin **stdout** -file --opt **--help** **--version**

`yum [`*параметр(ы)*`] [`*пакет(ы)*`]`

`yum` — это старый менеджер пакетов для пакетов RPM (файлы `.rpm`), который был заменен менеджером `dnf`. В следующей таблице перечислены общие операции с `yum`. Для операций с локальными файлами, которые `yum` не предоставляет, используйте команду `rpm`.

Операция	Команда yum
Поиск необходимого пакета (поддерживаются подстановочные знаки * и ?)	`yum search` *запрос*
Проверка того, установлен ли пакет	`yum list installed` *пакет*
Загрузка пакета без установки*	`sudo yum --downloadonly install` *пакет*

Операция	Команда yum
Загрузка пакета с установкой	sudo yum install *пакет*
Установка файла пакета	rpm -ivh *файл*.rpm
Просмотр информации о пакете	yum info *пакет*
Вывод содержимого пакета	rpm -ql *пакет*
Просмотр того, к какому пакету относится установленный файл	yum provides */путь/к/файлу*
Обновление установленного пакета	sudo yum update *пакет*
Удаление установленного пакета	sudo yum remove *пакет*
Перечисление всех пакетов, установленных в системе (совет: запускать в конвейере с командой less)	yum list installed
Проверка обновлений для установленных пакетов	yum check-update
Обновление всех пакетов в системе	sudo yum update

* Может потребоваться плагин downloadonly. Чтобы установить его, выполните команду sudo yum install yum-downloadonly.

rpm

stdin **stdout** -file --opt **--help** **--version**

rpm [*параметр(ы)*] [*файл(ы)*]

Если вы предпочитаете загружать и устанавливать пакеты RPM вручную, используйте команду rpm. В отличие от команд dnf или yum, rpm работает локально на вашем компьютере и не ищет новые пакеты в репозиториях программного обеспечения в интернете.

Имена файлов RPM обычно имеют следующий вид: *имя-версия.архитектура*.rpm. Например, имя файла emacs-29.1-2.x86_64.rpm указывает на пакет emacs версии 29.1-2 для 64-битных процессоров x86. Имейте в виду, что иногда rpm требует передать имя *файла* (например, emacs-29.1-2.x86_64.rpm), а иногда только имя пакета (например, emacs). В следующей таблице перечислены общие операции:

Операция	Команда rpm
Проверка того, установлен ли пакет	`rpm -q` *пакет*
Установка файла пакета	`sudo rpm -ivh` *файл*`.rpm`
Просмотр информации о пакете	`rpm -qi` *пакет*
Вывод содержимого пакета	`rpm -ql` *пакет*
Просмотр того, к какому пакету относится установленный файл	`rpm -qf` */путь/к/файлу*
Обновление установленного пакета	`sudo rpm -Uvh` *пакет*`.rpm`
Удаление установленного пакета	`sudo rpm -e` *пакет*
Перечисление всех пакетов, установленных в системе (совет: запускать в конвейере с командой `less`)	`rpm -qa`

APT

stdin **stdout** -file --opt **--help** **--version**

```
apt команда [параметр(ы)] пакет(ы)
dpkg [параметр(ы)] пакет(ы)
```

Набор команд APT (Advanced Packaging Tool) позволяет устанавливать и удалять пакеты Debian Linux (`.deb`), а также управлять ими. В следующей таблице перечислены общие операции.

Операция	Команда APT
(Перед выполнением других команд) Получение актуальной информации о доступных пакетах	`sudo apt update`
Поиск необходимого пакета	`apt search` *запрос*
Проверка того, установлен ли пакет	`apt policy` *пакет*
Загрузка пакета без установки	`sudo apt install -d` *пакет*
Загрузка пакета с установкой	`sudo apt install` *пакет*
Установка файла пакета	`sudo apt install` *файл*`.deb`
Просмотр информации о пакете	`apt show` *пакет*
Вывод содержимого пакета	`dpkg -L` *пакет*
Просмотр того, к какому пакету относится установленный файл	`dpkg -S` */путь/к/файлу*

Операция	Команда APT
Обновление установленного пакета	`sudo apt upgrade` *пакет*
Удаление установленного пакета	`sudo apt remove` *пакет*
Удаление установленного пакета и связанных с ним файлов	`sudo apt purge` *пакет*
Перечисление всех пакетов, установленных в системе (совет: запускать в конвейере с командой `less`)	`apt list --installed`
Проверка обновлений для установленных пакетов (сначала выполняется команда `sudo apt update`)	`sudo apt list --upgradable`
Обновление всех пакетов в системе	`sudo apt upgrade`
Обновление дистрибутива для следующей версии	`sudo apt dist-upgrade`

aptitude

stdin **stdout** -file --opt **--help** **--version**

`aptitude [`*параметр(ы)*`] [`*пакет(ы)*`]`

`aptitude` — это еще один менеджер пакетов для командной строки, который работает с пакетами Debian (`.deb`). При запуске без аргументов предоставляет полноценный интерактивный интерфейс для доступа к системе APT:

```
→ sudo aptitude
```

`aptitude` может выполнять и некоторые операции APT из командной строки. В следующей таблице перечислены общие операции, включая те, которые `aptitude` не поддерживает, и соответствующие команды `apt` или `dpkg`, которые следует запустить вместо них.

Операция	Команда aptitude
(Перед выполнением других команд) Получение актуальной информации о доступных пакетах	`sudo aptitude update`
Поиск необходимого пакета	`aptitude search` *запрос*
Проверка того, установлен ли пакет (см. столбец `State` в выводе)	`aptitude show` *пакет*
Загрузка пакета без установки	`aptitude download` *пакет*

Операция	Команда aptitude
Загрузка пакета с установкой	sudo aptitude install *пакет*
Установка файла пакета	sudo apt install *файл*.deb
Просмотр информации о пакете	aptitude show *пакет*
Вывод содержимого пакета	dpkg -L *пакет*
Просмотр того, к какому пакету относится установленный файл	dpkg -S */путь/к/файлу*
Обновление установленного пакета	sudo aptitude safe-upgrade *пакет*
Удаление установленного пакета	sudo aptitude remove *пакет*
Перечисление всех пакетов, установленных в системе (совет: запускать в конвейере с командой less)	aptitude search ~i
Проверка обновлений для установленных пакетов	aptitude --simulate full-upgrade
Обновление всех пакетов в системе	sudo aptitude full-upgrade

emerge

stdin **stdout** -file --opt **--help** **--version**

```
emerge [параметр(ы)] [аргумент(ы)]
emaint [параметр(ы)] команда
equery [параметр(ы)] команда [аргумент(ы)]
```

Команда emerge управляет менеджером пакетов Portage в Gentoo Linux. Перед началом работы с пакетами Portage выполните следующую команду:

```
→ sudo emerge gentoolkit        Установка дополнительных инструментов Portage
```

Операция	Команда emerge
(Перед выполнением других команд) Получение актуальной информации о доступных пакетах	sudo emaint -a sync
Поиск необходимого пакета (по имени)	emerge -s *запрос*
Поиск необходимого пакета (по описанию)	emerge -S *запрос*
Проверка того, установлен ли пакет	equery list "*" | grep *пакет*

Операция	Команда emerge
Загрузка пакета без установки	`sudo emerge -f` *пакет*
Загрузка пакета с установкой	`sudo emerge` *пакет*
Просмотр информации о пакете	`sudo equery meta [--description]` *пакет*
Вывод содержимого пакета	`equery files` *пакет*
Просмотр того, к какому пакету относится установленный файл	`equery belongs` */путь/к/файлу*
Обновление установленного пакета	`sudo emerge -u` *пакет*
Удаление установленного пакета	`sudo emerge -cav` *пакет*
Перечисление всех пакетов, установленных в системе (совет: запускать в конвейере с командой `less`)	`equery list "*"`
Проверка обновлений для установленных пакетов	`emerge -puD world`
Обновление всех пакетов в системе	`sudo emerge -uD world`

Если команда `emerge` не удаляет пакет из-за зависимостей, то вы можете указать это в командной строке:

```
→ sudo emerge -cav my/package                          Неудача
Calculating dependencies... done!
  my/package-29.3 pulled in by:
    other/pack-16.1 requires ...                       Зависимость!
>>> No packages selected for removal by depclean
→ sudo emerge -cav my/package other/pack               Успех
Would you like to unmerge these packages? [Yes/No] Yes
```

pacman

stdin **stdout** -file --opt **--help** **--version**

```
pacman команда [параметр(ы)] [аргумент(ы)]
```

Команда `pacman` — менеджер пакетов для Arch Linux. Обычно пакеты Arch представляют собой `tar`-файлы, сжатые с помощью команды `zstd`. В следующей таблице перечислены поддерживаемые операции.

Операция	Команда pacman
(Перед выполнением других команд) Получение актуальной информации о доступных пакетах	`sudo pacman -Sy`
Поиск необходимого пакета (по шаблону)	`pacman -Ss` *запрос*
Проверка того, установлен ли пакет	`pacman -Q` *пакет*
Загрузка пакета без установки	`sudo pacman -Sw` *пакет*
Загрузка пакета с установкой	`sudo pacman -S` *пакет*
Установка файла пакета	`sudo pacman -U` *файл*`.pkg.tar.zstm`
Просмотр информации о пакете	`pacman -Qi` *пакет*
Вывод содержимого пакета	`pacman -Ql` *пакет*
Просмотр того, к какому пакету относится установленный файл	`pacman -Qo` */путь/к/файлу*
Обновление установленного пакета	`sudo pacman -S` *пакет*
Удаление установленного пакета	`sudo pacman -R` *пакет*
Перечисление всех пакетов, установленных в системе (совет: запускать в конвейере с командой `less`)	`pacman -Qe`
Проверка обновлений для установленных пакетов	`sudo pacman -Syup`
Обновление всех пакетов в системе	`sudo pacman -Syu`

zypper

stdin **stdout** -file --opt **--help** **--version**

`zypper [`*параметр(ы)*`] `*команда*` [`*параметр(ы)_команды*`] [`*аргумент(ы)*`]`

Команда `zypper` — это менеджер пакетов для openSUSE Linux. Команда оперирует пакетами RPM. В следующей таблице перечислены поддерживаемые операции.

Операция	Команда zypper
(Перед выполнением других команд) Получение актуальной информации о доступных пакетах	`sudo zypper refresh`
Поиск необходимого пакета	`zypper search` *запрос*
Проверка того, установлен ли пакет	`zypper search -i` *пакет*

Операция	Команда zypper
Загрузка пакета без установки	`sudo zypper install -d` *пакет*
Загрузка пакета с установкой	`sudo zypper install` *пакет*
Установка файла пакета	`sudo rpm -ivh` *файл*`.rpm`
Просмотр информации о пакете	`zypper info` *пакет*
Вывод содержимого пакета	`rpm -ql` *пакет*
Просмотр того, к какому пакету относится установленный файл	`rpm -qf` */путь/к/файлу*
Обновление установленного пакета	`sudo zypper update` *пакет*
Удаление установленного пакета	`sudo zypper remove` *пакет*
Перечисление всех пакетов, установленных в системе (совет: запускать в конвейере с командой `less`)	`zypper search -i`
Проверка обновлений для установленных пакетов	`sudo zypper update --dry-run`
Обновление всех пакетов в системе	`sudo zypper update`
Обновление дистрибутива для следующей версии	`sudo zypper dist-upgrade`

flatpak

stdin **stdout** -file --opt **--help** **--version**

`flatpak` *команда* [*параметр(ы)*] [*аргумент(ы)*]

Flatpak — это система для установки пакетов ПО типа Flatpak. Она запускается в ограниченной среде, называемой *контейнером*. Контейнер включает в себя все зависимые компоненты пакета. Для установки, обновления и удаления пакетов используйте команду `flatpak`. Возможно, сначала потребуется добавить сетевой репозиторий:

```
→ sudo flatpak remote-add --if-not-exists flathub \
https://flathub.org/repo/flathub.flatpakrepo
```

Система Flatpak имеет ограниченный доступ к файловой системе хоста, но по большей части работает как обычное приложение... За исключением случаев, когда вы запускаете команду из командной строки. Система должна запускаться

с помощью команды `flatpak`. Если вы установили GNU Emacs, узнайте соответствующий идентификатор Flatpak, представляющий собой трехкомпонентную строку с точками, например `org.gnu.emacs`. Затем запустите программу по ее идентификатору:

```
→ flatpak list | grep emacs
ID              ...
org.gnu.emacs ...                                    Идентификатор flatpak
→ flatpak run org.gnu.emacs
```

Если вам не хочется вводить идентификатор, можно определить псевдоним:

```
alias emacs='flatpak run org.gnu.emacs'
```

В следующей таблице перечислены общие операции для работы с Flatpak в рамках всей системы (для этого часто нужны права суперпользователя). Чтобы установить Flatpaks только для себя, выполните команду `flatpak --user` вместо `sudo flatpak`.

Операция	Команда flatpak
Добавление сетевого репозитория для загрузки Flatpaks	`sudo flatpak remote-add --if-not-exists` *`имя url-адрес`*
Перечисление сетевых репозиториев, добавленных в систему	`flatpak` *`репозиторий`*
Поиск необходимого пакета	`flatpak search` *`запрос`*
Проверка того, установлен ли пакет	`flatpak list | grep` *`пакет`*
Загрузка пакета с установкой	`sudo flatpak install` *`пакет`*
Установка файла пакета	`sudo flatpak install` *`/путь/к/файлу`*
Запуск пакета	`flatpak run` *`идентификатор`*
Просмотр информации о пакете	`flatpak info` *`идентификатор`*
Обновление установленного пакета	`sudo flatpak update` *`идентификатор`*
Удаление установленного пакета	`sudo flatpak uninstall` *`идентификатор`*
Перечисление всех пакетов, установленных в системе (совет: запускать в конвейере с командой `less`)	`flatpak list`
Обновление всех пакетов в системе	`sudo flatpak update`

snap

stdin **stdout** -file --opt **--help** **--version**

```
snap [параметр(ы)] команда [параметр(ы)_команды]
```

Snap — это система установки пакетов ПО. Она запускается в ограниченной среде, называемой *контейнером* или *песочницей*. Контейнер включает в себя все зависимости пакета. Пакеты Snap имеют ограниченный доступ к файловой системе хоста, но чаще всего функционируют как обычные приложения. Используйте команду `snap` для установки, обновления и удаления пакетов. В следующей таблице перечислены поддерживаемые операции.

Операция	Команда snap
Поиск необходимого пакета	`snap find` *запрос*
Проверка того, установлен ли пакет	`snap list | grep` *пакет*
Загрузка пакета без установки	`sudo snap download` *пакет*
Загрузка пакета с установкой	`sudo snap install` *пакет*
Просмотр информации о пакете	`snap info` *пакет*
Вывод содержимого пакета	`ls /snap/` *пакет* `/current`
Обновление установленного пакета	`sudo snap refresh` *пакет*
Удаление установленного пакета	`sudo snap remove` *пакет*
Перечисление всех пакетов, установленных в системе (совет: запускать в конвейере с командой `less`)	`snap list`
Проверка обновлений для установленных пакетов	`snap refresh --list`
Обновление всех пакетов в системе	`sudo snap refresh`

Компилирование и установка ПО

`configure`	Подготовка к компиляции ПО с помощью `make`
`make`	Создание ПО из исходной программы

Менеджеры пакетов, описанные в разделе «Установка ПО» данной главы, для установки программ требуют права суперпользователя. Будучи обычным пользователем, вы

тоже можете установить ПО в домашнем каталоге. Для этого требуется сделать больше шагов и лучше понимать процесс, чем когда вы просто задействуете менеджер пакетов.

Загрузка файлов исходного кода

Многие программы для Linux распространяются в виде файлов *исходного кода*, которые можно загрузить в вашу локальную систему. Наиболее популярны следующие два способа распространения:

- сжатые файлы TAR или ZIP, опубликованные на различных веб-сайтах;
- Git-репозитории, размещенные на серверах GitHub.

Я кратко рассажу о каждом из способов, а затем покажу, как скомпилировать исходный код с помощью команд `configure` и `make`.

Метод 1. Скачивание и распаковка TAR- или ZIP-файла

Архив TAR — это набор файлов, упакованный с помощью программы `tar`. Обычно файл сжимается с помощью команды `gzip` (его расширение `.tar.gz` или `.tgz`) или команды `bzip2` (расширение `.tar.bz2` или `.tbz`). Аналогично файл `.zip` — это набор файлов, упакованных с помощью команды `zip`. Чтобы распаковать файлы, сделайте следующее.

1. Просмотрите содержимое пакета. Убедитесь, что при извлечении оно не перезапишет существующие файлы в вашей системе, случайно или по злому умыслу[1]:

```
→ tar -tvf package.tar.gz | less          gzip
→ tar -tvf package.tar.bz2 | less         bzip2
→ unzip -l package.zip | less             zip
```

[1] Вредоносный архив может содержать абсолютный путь к файлу, например /etc/passwd, который после извлечения способен перезаписать ваш системный файл паролей. Очень неприятная ситуация.

2. Если вас все устраивает, распакуйте файлы в новый каталог:

```
→ mkdir newdir
→ tar -xvf package.tar.gz -C newdir        gzip
→ tar -xvf package.tar.bz2 -C newdir       bzip2
→ unzip -d newdir package.zip | less       zip
```

Если все сработало, перейдите к разделу «Компиляция и запуск кода» далее в этой главе.

Метод 2. Клонирование Git-репозитория

Чтобы скачать ПО из GitHub или другого подобного хранилища, скопируйте URL-адрес нужного хранилища и передайте его команде `git clone`. Команда должна выглядеть примерно так:

```
→ git clone git@github.com:пользователь/репозиторий.git
```

`git clone` загружает копию репозитория в вашу локальную систему. Теперь система готова к компиляции программы.

Компиляция и запуск кода

После того как вы загрузили и/или извлекли исходный код, скомпилируйте программу. Вкратце обычная последовательность команд такова:

```
→ ./configure PREFIX=каталог_с_возможностью_записи
→ make
→ make install
```

Если подробнее, то последовательность действий выглядит следующим образом.

1. В каталоге с файлами исходного кода прочитайте распакованный файл с именем `INSTALL` или `README`, например, так:

```
→ less INSTALL
```

2. Обычно сначала нужно запустить сценарий `configure`, затем процедуру `make`, затем выполнить команду `make install`. Чтобы посмотреть параметры сценария

`configure`, выполните следующую команду (обычно вывод очень длинный):

```
→ ./configure --help | less
```

Наиболее важным параметром является `--prefix`, определяющий каталог установки. Например, чтобы установить программу в подкаталог `packages` домашнего каталога, вы должны выполнить команду

```
→ ./configure --prefix=$HOME/packages
```

Если вы опустите этот параметр, то `configure` организует установку ПО в масштабах всей системы и для завершения вам понадобятся права суперпользователя:

```
→ ./configure
```

Если `configure` не работает, то в вашей локальной системе не хватает какого-то необходимого ПО. Внимательно прочитайте вывод `configure`, установите недостающее ПО и повторите попытку.

3. После успешного выполнения `configure` выполните компиляцию (сборку) ПО, запустив команду `make`:

```
→ make
```

Она выполнит все необходимые действия по подготовке ПО к установке, при этом не устанавливая его. Если команда `make` не работает, внимательно прочитайте сообщение об ошибке, поищите решения в интернете и при необходимости напишите создателям программы об ошибке.

4. Если команда `make` была успешно выполнена и вы запустили `configure` с параметром `--prefix`, то завершите установку:

```
→ make install
```

А если вы опустили параметр, чтобы установить ПО в свою систему, используйте команду `sudo`:

```
→ sudo make install
```

Если вы изначально запускали `configure` с параметром `--prefix`, установленное программное обеспечение будет находиться в указанном вами каталоге, обычно в подкаталоге с именем `bin`. По желанию добавьте этот каталог в путь поиска (см. раздел «Маршрут поиска» главы 1). Если вы установили программу в каталог `$HOME/packages`, как в моем примере, то добавьте ее подкаталог `bin` в `PATH` с помощью следующей команды:

```
→ PATH=$PATH:$HOME/packages/bin
```

Добавьте эту строку в файл конфигурации оболочки (см. раздел «Настройка оболочки» главы 1), чтобы новое ПО стало доступно.

ГЛАВА 4

Обслуживание файловой системы

Использование дисков и файловых систем

`df`	Отображение доступного пространства на смонтированных файловых системах
`lsblk`	Перечисление дисков и других блочных устройств
`mount`	Осуществление доступа (монтирование) к разделу диска
`umount`	Размонтирование раздела диска (прекращение доступа)
`fsck`	Проверка раздела диска на отсутствие ошибок

В системах Linux могут использоваться несколько дисков или разделов. Обычно их называют устройствами, файловыми системами, томами и даже каталогами. Я же постараюсь быть более точным терминологически.

Диск — это устройство хранения данных, которое может быть разделено на *разделы*, выступающие в качестве независимых устройств. Диски и разделы представлены в системах Linux в виде специальных файлов в каталоге `/dev`. Например, `/dev/sda7` может быть разделом вашего жесткого диска. К числу распространенных устройств в каталоге `/dev` относятся следующие:

`sda`	Первое блочное устройство, например жесткие диски SCSI, SATA или USB; разделы `sda1`, `sda2`
`sdb`	Второе блочное устройство; разделы `sdb1`, `sdb2`… Аналогично для `sdc`, `sdd` и т. д.
`md0`	Первое устройство RAID; разделы `md0p1`, `md0p2`… Аналогично для `md1`, `md2` и т. д.
`nvme0n1`	Первое устройство NVMe SSD; разделы `nvme0n1p1`, `nvme0n1p2`… Аналогично для `nvme1n1`, `nvme2n1` и т. д. Второе целое число, как и 1 в `nvme0n1p2`, называется *идентификатором пространства имен*, большинство пользователей могут не обращать на него внимания

Прежде чем хранить файлы в разделе, его нужно отформатировать. Затем в нем создается *файловая система* (см. раздел «Создание и изменение файловых систем» далее в этой главе). Она определяет способ предоставления файлов. Примерами являются ext4 (журналируемая файловая система Linux) и NTFS (файловая система Microsoft Windows). Обычно форматирование выполняется при установке Linux.

После создания файловую систему можно сделать доступной, *смонтировав* ее раздел как пустой каталог[1]. Например, если вы смонтируете файловую систему Windows как каталог `/mnt/win`, то она станет частью дерева каталогов вашей системы и вы сможете создавать и редактировать файлы типа `/mnt/win/myfile.txt`. Обычно монтирование происходит автоматически во время загрузки. Чтобы сделать разделы недоступными через файловую систему, следует *размонтировать* их.

df

stdin **stdout** -file **--opt** **--help** **--version**

`df [`*`параметр(ы)`*`] [`*`диск(и)`*` | `*`файл(ы)`*` | `*`каталог(и)`*`]`

Команда `df` показывает размер, используемое и свободное пространство на выбранном разделе *диска*. Если вы укажете *каталог* или *файл*, `df` выведет информацию о дисковом устройстве, на котором находится объект. Без аргументов `df` сообщает обо всех файловых системах:

```
→ df
Filesystem 1k-blocks    Used   Avail Use% Mounted on
/dev/sda     1011928  225464  735060  24% /
/dev/sda9     521748  249148  246096  51% /var
/dev/sda8    8064272 4088636 3565984  54% /usr
/dev/sda10   8064272 4586576 3068044  60% /home
```

Команда `df` может перечислять не только диски, но и другие устройства. Чтобы сократить вывод и вывести только диски,

[1] Можно смонтировать файловую систему в каталог с другими файлами, но содержимое каталога будет недоступным до тех пор, пока вы не размонтируете его.

попробуйте указать следующие параметры (и создайте псевдоним, если необходимо):

```
→ df -h -x tmpfs -x devtmpfs -x squashfs
```

Полезные параметры

`-k`	Вывод размера в килобайтах
`-m`	Вывод размера в мегабайтах
`-B` *N*	Вывод размера в блоках по *N* байт (по умолчанию — 1024)
`-h`	Вывод размера в единицах, понятных человеку (следует выбрать под-
`-H`	ходящую единицу изменения для каждого размера). Например, если на двух дисках свободно 1 Гбайт и 25 Кбайт соответственно, то `df -h` выведет 1G и 25K. Параметр `-h` использует значения 1024, а `-H` — 1000
`-l`	Вывод только локальных файловых систем
`-T`	Дополнение вывода типами файловых систем (`ext3`, `vfat` и т. д.)
`-t` *тип*	Вывод файловых систем только указанного *типа*
`-x` *тип*	Без вывода файловых систем указанного *типа*
`-i`	Режим индексного дескриптора. Отображение общего количества индексных дескрипторов (использованных и свободных) для каждой файловой системы вместо дисковых блоков. Когда применены все индексные дескрипторы файловой системы, система считается заполненной, даже если на диске есть свободное место

lsblk

stdin **stdout** -file **--opt** **--help** **--version**

```
lsblk [параметр(ы)] [устройство(а)]
```

Команда `lsblk` перечисляет устройства хранения данных (блочные устройства) Linux, такие как жесткие диски, SSD и RAM-диски:

```
→ lsblk
NAME     MAJ:MIN RM   SIZE RO TYPE  MOUNTPOINTS
sda        8:0    0    20G  0 disk
├─sda1     8:1    0     1M  0 part
├─sda2     8:2    0   513M  0 part  /boot/efi
├─sda3     8:3    0  19.5G  0 part  /
sdb        8:80   0   7.6G  0 disk
└─sdb1     8:81   1   7.6G  0 part  /mnt/usb-key
```

В выводе показаны жесткий диск по адресу `/dev/sda` с тремя разделами и USB-флешка по адресу `/dev/sdb` с одним

разделом. Команда имеет множество параметров форматирования, вы можете сократить вывод команды, отобразив только нужные устройства:

```
→ lsblk -o NAME,SIZE /dev/sda
NAME    SIZE
sda      20G
├─sda1    1M
├─sda2  513M
└─sda3 19.5G
```

Полезные параметры

`-l`	Вывод простого списка вместо дерева
`-a`	Вывод всех блочных устройств, включая скрытые
`-f`	Добавление в вывод информации о файловых системах на устройствах
`-o` *столбцы*	Вывод только определенных *столбцов* в виде разделенного запятыми списка. Просмотр столбцов доступен с помощью команды `lsblk --help`
`-J`	Вывод списка в формате JSON для обработки другими программами

mount

stdin **stdout** -file **--opt** **--help** **--version**

```
mount [параметр(ы)] [устройство | каталог]
```

Команда `mount` реализует доступ к разделу. Чаще всего она используется с дисковыми накопителями (например, `/dev/sda1`) и съемными носителями (например, USB-носителями), осуществляя доступ к ним через существующий каталог (допустим, `/mnt/mydir`):

```
→ sudo mkdir /mnt/mydir
→ ls /mnt/mydir                    Обратите внимание, что раздел пуст
→ sudo mount /dev/sda1 /mnt/mydir
→ ls /mnt/mydir
file1  file2  file3                Файлы в смонтированном разделе
→ df /mnt/mydir
Filesystem 1K-blocks   Used  Avail Use% Mounted on
/dev/sda1    1011928 285744 674780  30% /mnt/mydir
```

Команда `mount` имеет множество применений, я рассматриваю только основные функции. В большинстве случаев `mount` считывает файл `/etc/fstab` (filesystem table — «таблица

файловой системы»), извлекая из него информацию о способе монтирования нужного диска. Например, если вы выполните команду `mount/usr`, команда `mount` ищет в файле `/etc/fstab` строку `/usr`, которая может выглядеть следующим образом:

```
/dev/sda8    /usr    ext4    defaults    1    2
```

В данном случае команда `mount` сообщает, что устройство `/dev/sda8` должно быть смонтировано в `/usr` как файловая система (ext4) Linux с параметрами по умолчанию. Смонтируйте его с помощью одной из этих команд:

```
→ sudo mount /dev/sda8          Как устройство
→ sudo mount /usr               Как каталог
```

Чаще всего команда `mount` выполняется суперпользователем, но обычные съемные устройства, например USB-носители и DVD, могут быть смонтированы/размонтированы любыми пользователями.

Полезные параметры

`-t` *`тип`*	Указание *`типа`* файловой системы, например `ext4` или `ntfs`
`-l`	Вывод списка всех смонтированных файловых систем, работает и с `-t`
`-a`	Монтирование всех файловых систем, перечисленных в `/etc/fstab`. Игнорируются записи с параметром `noauto`. Работает и с `-t`
`-r`	Монтирование файловой системы только для чтения (ищите в документации информацию об ограничениях)

umount

stdin **stdout** -file **--opt** **--help** **--version**

```
umount [параметр(ы)] [устройство | каталог]
```

Функция `umount` противоположна команде `mount`: она прекращает доступ к разделу для поиска через файловую систему[1]. Например, если вы смонтировали флешку, то прежде чем вытащить из разъема, нужно размонтировать ее:

```
→ umount "/media/smith/My Vacation Photos"
```

[1] Обратите внимание на написание: umount, а не unmount!

Перед извлечением всегда нужно размонтировать все съемные носители, иначе вы рискуете вывести из строя их файловую систему. Чтобы размонтировать все смонтированные устройства, выполните команду

```
→ sudo umount -a
```

Осторожно: не размонтируйте используемую файловую систему! Тут стоит сказать, что `umount` откажется вам повиноваться из соображений безопасности.

fsck

stdin **stdout** -file **--opt** **--help** **--version**

```
fsck [параметр(ы)] [устройство(а)]
```

Команда `fsck` (filesystem check) проверяет файловую систему диска Linux и при необходимости исправляет найденные ошибки. Она запускается автоматически при загрузке системы или вручную. Как правило, перед проверкой необходимо размонтировать устройство, чтобы на нем не были запущены какие-либо программы:

```
→ sudo umount /dev/sda10
→ sudo fsck -f /dev/sda10
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information
/home: 172/1281696 files (11.6% non-contiguous), ...
```

Вы не можете использовать `fsck` для исправления корневой файловой системы, если система работает нормально. Сначала нужно загрузиться с USB-носителя Linux или другого устройства восстановления, а затем выполнить команду `fsck`.

`fsck` — это клиентская часть для набора команд проверки файловой системы, находящихся в `/sbin` и имеющих имена, начинающиеся с «fsck». Поддерживаются только определенные типы файловых систем, перечислить их можно с помощью следующей команды:

```
→ ls /sbin/fsck.* | cut -d. -f2 | column
cramfs      ext3        fat         hfsplus      msdos
ext2        ext4        hfs         minix        vfat
```

Полезные параметры

`-A` Проверка по порядку всех дисков, перечисленных в `/etc/fstab`

`-f` Принудительный запуск `fsck`, в том числе без явных ошибок

`-N` Вывод описания запланированной проверки, а затем выход без ее выполнения

`-r` Исправление ошибок в интерактивном режиме с уведомлением перед каждым исправлением

`-a` Исправление ошибок автоматически (используйте только в том случае, если вы *уверены*, что делаете, в противном случае можете вывести из строя файловую систему)

Создание и изменение файловых систем

`mkfs`	Форматирование (создание файловой системы) раздела диска
`resize2fs`	Увеличение/уменьшение раздела диска
`e2label`	Изменение метки тома раздела диска

Операции с дисками, например форматирование, иногда трудно выполнить в командной строке. Для более сложных, чем форматирование одного раздела, операций я рекомендую использовать графическое приложение `grapted`. Честно говоря, работать в нем намного проще и меньше вероятность допустить ошибку.

Тем не менее я по-прежнему выполняю простые команды в командной строке. Одна из них — перечисление разделов устройства хранения данных типа `/dev/sda` с помощью `fdisk`:

```
→ sudo fdisk -l /dev/sda
Disk /dev/sda: 20 GiB, 21474836480 bytes, ...
⋮
Device       Start       End   Sectors   Size Type
/dev/sda1     2048      4095      2048     1M BIOS boot
/dev/sda2     4096   1054719   1050624   513M EFI System
/dev/sda3  1054720  41940991  40886272  19.5G Linux
```

Или аналогично командой `parted`:

```
→ sudo parted /dev/sda — print
```

Еще один способ — экспорт таблицы разделов устройства хранения данных для безопасности файлов (храните таблицу на USB-носителе или другом устройстве, а не на рабочем диске!):

```
→ sudo sfdisk -d /dev/sda > /mnt/thumb/sda.txt
```

Если вы допустите ошибку в ответственный момент, то сможете восстановить таблицу разделов (но будьте осторожны, указывая дисковое устройство, иначе ошибочно перезапишете неправильную таблицу разделов):

```
→ sudo sfdisk /dev/device < /mnt/thumb/sda.txt
```

Перечисленные далее команды относятся к базовым операциям с дисками и файловыми системами без использования графических инструментов.

mkfs

stdin **stdout** -file --opt **--help** **--version**

```
mke2fs [параметр(ы)] устройство
mkfs.ext3 [параметр(ы)] устройство
mkfs.ext4 [параметр(ы)] устройство
mkntfs [параметр(ы)] устройство
mkfs.ntfs [параметр(ы)] устройство      ...и много других вариантов...
```

Семейство команд `mkfs` форматирует устройство хранения данных Linux с применением различных файловых систем. Устройство хранения обычно представляет собой раздел, например `/dev/sdb1`.

ВНИМАНИЕ

Команда mfsk форматирует устройство хранения данных. Убедитесь, что вы правильно указали имя устройства, иначе рискуете потерять данные!

Примеры:

```
→ sudo mkfs.ext4 /dev/device      Стандартная файловая система Linux
→ sudo mke2fs /dev/device         Стандартная файловая система Linux
→ sudo mkfs.ntfs /dev/device      Файловая система Microsoft Windows
→ sudo mkntfs /dev/device         Файловая система Microsoft Windows
```

Итак, в качестве имен многих команд используется слово mkfs, после которого указывается точка и тип файловой системы, например `mkfs.ext4`. У них могут быть альтернативные имена (ссылки) с типом файловой системы, находящимся внутри слова mkfs, например `mke2fs` для файловой системы ext. Чтобы перечислить все подобные команды в своей системе, выполните следующую команду:

```
→ ls /usr/*bin/mkfs.*
/usr/sbin/mkfs.ext2     /usr/sbin/mkfs.hfs
/usr/sbin/mkfs.ext3     /usr/sbin/mkfs.minix
/usr/sbin/mkfs.ext4     /usr/sbin/mkfs.msdos
/usr/sbin/mkfs.fat      /usr/sbin/mkfs.ntfs
```

Полезные параметры

`-n` Режим формального прогона — без фактического форматирования. Вывод *предполагаемых* задач

`-L` *имя* Присвоение тому после форматирования указанного *имени* длиной не более 16 байт

`-b` *N* Использование блоков размером *N* байт

resize2fs

stdin stdout -file --opt --help --version

```
resize2fs [параметр(ы)] устройство [размер]
```

Команда `resize2fs` увеличивает/уменьшает (расширяет/сокращает) стандартную файловую систему Linux типа ext2, ext3 или ext4. Увеличить файловую систему можно так.

1. Убедитесь, что на устройстве достаточно свободного пространства.
2. Размонтируйте файловую систему.
3. Увеличьте раздел диска с помощью команды `gparted` или аналогичной программы. (Для этого необходимо свободное пространство на диске.)
4. Проверьте файловую систему с помощью `fsck`.
5. Запустите команду `resize2fs` с соответствующими аргументами. В современных версиях ОС файловая система может быть смонтирована во время изменения размера.

Чтобы уменьшить размер файловой системы, выполните следующее.

1. С помощью команды `df` проверьте, что данные в файловой системе (столбец `Used`) вписываются в предложенный новый размер.
2. Размонтируйте файловую систему.
3. Запустите команду `resize2fs` с соответствующими аргументами.
4. Уменьшите раздел диска с помощью команды `gparted` или аналогичной программы.
5. Проверьте файловую систему с помощью команды `fsck`.

Чтобы изменить размер файловой системы `/dev/sda1` (предполагается, что вы уже завершили проверку), запустите команду `resize2fs` и укажите размер (если необходимо):

```
→ sudo resize2fs /dev/sda1 100G      Изменение размера до 100 Гбайт
→ sudo resize2fs /dev/sda1           Изменение размера
```

Размер может быть представлен абсолютным количеством блоков, например `12345690`, или цифровым значением с буквой (`K` — Кбайт, `M` — Мбайт, `G` — Гбайт, `T` — Тбайт или `s` — 512-байтный сектор). Значения являются степенями двойки, поэтому `1K` означает 1024, а не 1000, и т. д.

Если вы часто изменяете размеры файловых систем, то удобнее управлять логическими томами (LVM). Подробнее об этом говорится в разделе «Хранение данных в системе LVM» немного позже. Вы также можете использовать более современную файловую систему, описанную в разделе «ZFS — современная универсальная файловая система» далее в этой главе.

Полезные параметры

`-f` Принудительное изменение размера, даже если команда `resize2fs` выдает предупреждения

`-p` Отображение хода выполнения операции

e2label

stdin stdout -file --opt --help --version

```
e2label устройство [метка]
```

Метка — это псевдоним файловой системы. Команда `e2label` присваивает или отображает метку стандартной файловой системы Linux типа ext2, ext3 или ext4. Присваивать метки файловым системам не требуется, но они удобны при использовании ссылок на файловые системы в `/etc/fstab`:

```
→ sudo e2label /dev/sdb1 backups          Присвоить ярлык
→ sudo e2label /dev/sdb1                  Вывести ярлыки
backups
```

RAID-массивы для резервирования

`mdadm` Управление массивами RAID

RAID (Redundant Array of Independent Disks — избыточный массив независимых (самостоятельных) дисков) — это технология, которая распределяет данные компьютера по нескольким дискам, при этом несколько дисков действуют как один цельный диск. Обычно массивы RAID служат для обеспечения сохранности данных — если один диск перестанет работать, копии файлов сохранятся на оставшихся. Другие типы RAID повышают производительность системы хранения данных.

Группа объединенных дисков называется *массивом* RAID. Тип RAID-массива, называемый *уровнем* RAID, определяет количество отказов, которое может выдержать массив. Существуют стандартные уровни RAID: RAID-0, RAID-1, RAID-5, RAID-10 и др., о которых вы можете прочитать в интернете.

Давайте создадим массив RAID-1 с помощью самой распространенной программы для управления RAID — `mdadm`. RAID-1 добавляет избыточность, просто зеркалируя данные с одного диска на другие. Пока один из дисков работает, данные в безопасности. В этом примере я начинаю с двух

дисков, /dev/sdf и /dev/sdg, на каждом из которых есть раздел размером 10 Гбайт, /dev/sdf1 и /dev/sdg1 соответственно. Шаги, которые я показываю, во многом аналогичны для RAID других уровней и дополнительных устройств. Для получения подробной информации посетите Linux Raid Wiki (https://oreil.ly/ibL7l).

ВНИМАНИЕ

Операции с RAID-массивами могут уничтожить файловые системы (при этом подтверждение не запрашивается). Для безопасности практикуйте команды на резервных дисках или виртуальной машине.

Создание RAID-массива

Сначала нужно сообщить системе, что RAID-массива еще не существует:

```
→ cat /proc/mdstat
Personalities :                    Типы RAID не перечислены
```

Теперь создайте массив RAID-1 /dev/md1, состоящий из двух разделов:

```
→ sudo mdadm --create /dev/md1 --level 1 \
  --raid-devices 2 /dev/sdf1 /dev/sdg1
```

Снова просмотрите содержимое /proc/mdstat. Теперь в строке Personalities указано, что используется RAID-1, а в следующей строке приведен новый массив, md1, который находится в процессе создания:

```
→ cat /proc/mdstat
Personalities : [raid1]
md1 : active raid1 sdg1[1] sdf1[0]
      10474496 blocks super 1.2 [2/2] [UU]
      [=========>.........]  resync = 45.8% ...
                             finish=0.4min  ...
```

Если хотите, можете дождаться завершения операции (resync):

```
→ cat /proc/mdstat
Personalities : [raid1]
md1 : active raid1 sdg1[1] sdf1[0]
      10474496 blocks super 1.2 [2/2] [UU]
```

Вам не придется долго ждать. Массив уже можно использовать. Отформатируйте и смонтируйте его, как любое другое устройство хранения данных:

```
→ sudo mke2fs /dev/md1                          Форматирование массива
→ sudo mkdir /mnt/raid                          Монтирование массива
→ sudo mount /dev/md1 /mnt/raid
→ df -h /mnt/raid                               Просмотр массива
Filesystem      Size  Used Avail Use% Mounted on
/dev/md1        9.9G   24K  9.4G   1% /mnt/raid
```

Запустите команду `lsblk`, чтобы отобразить конфигурацию RAID:

```
→ lsblk
⋮
sdf           8:80   0    10G  0 disk
└─sdf1        8:81   0    10G  0 part
  └─md1       9:1    0    10G  0 raid1 /mnt/raid
sdg           8:96   0    10G  0 disk
└─sdg1        8:97   0    10G  0 part
  └─md1       9:1    0    10G  0 raid1 /mnt/raid
```

Запустите команду `mdadm`, чтобы получить подробную информацию о массиве:

```
→ sudo mdadm --detail /dev/md1
dev/md1:
  ⋮
  Creation Time : Thu Jul 20 13:15:08 2023
     Raid Level : raid1
     Array Size : 10474496 (9.99 GiB 10.73 GB)
   Raid Devices : 2
          State : clean
Working Devices : 2
  ⋮
Number  Major  Minor  RaidDevice State
  0      8      81       0       active sync /dev/sdf1
  1      8      97       1       active sync /dev/sdg1
```

Когда массив будет готов, сохраните его конфигурацию, чтобы он монтировался автоматически при загрузке системы. **Не пропускайте ни одного шага!**

1. Сохраните конфигурацию в RAID-файл:

```
→ sudo mdadm --detail --scan --verbose > /tmp/raid
→ cat /tmp/raid
ARRAY /dev/md1 level=raid1 num-devices=2 ...
```

2. С помощью текстового редактора добавьте содержимое `/tmp/raid` в конфигурационный файл `/etc/mdadm/mdadm.conf`, заменив все предыдущие конфигурации RAID.
3. Выполните следующую команду, чтобы обновить ядро Linux:

```
→ sudo update-initramfs -u
```

4. Перезагрузите компьютер. Проверьте, что RAID-массив сохранился, смонтировав его вручную[1]:

```
→ sudo mount /dev/md1 /mnt/raid
```

5. Если все работает исправно, добавьте следующую строку в `/etc/fstab`, чтобы ваш RAID-массив монтировался во время загрузки:

```
/dev/md1  /mnt/raid  ext4  defaults  0  2
```

ВНИМАНИЕ!

Не обновляйте /etc/fstab слишком рано. Если в конфигурации RAID возникнут проблемы и вы перезагрузитесь, компьютер может зависнуть. Сначала проверьте конфигурацию, перезагрузившись и смонтировав массив вручную, как это сделал я.

---

[1] Если ваш RAID-массив по неизвестным причинам переименовывается в /dev/md127, то на предыдущем шаге необходимо выполнить команду update-initramfs.

Замена устройства в RAID-массиве

Итак, вы запустили свой массив и он исправно работает. А что происходит, когда устройство перестает работать и требует замены? Во-первых, вы увидите сбой в `/proc/mdstat`. Неработающее устройство, `/dev/sdf1`, помечено символом `(F)`, а индикатор времени работы, который должен считывать `[UU]` (два устройства работают), выводит `[U_]` (первое устройство работает, второе — нет):

```
→ cat /proc/mdstat
Personalities : [raid1]
md1 : active raid1 sdf1[1](F) sdg1[0]
      10474496 blocks super 1.2 [2/1] [U_]
```

Команда `mdadm` также отображает массив как некорректно работающий (`degraded`), а устройство — как неисправное (`faulty`):

```
→ sudo mdadm --detail /dev/md1
/dev/md1:
   ⋮
      Raid Devices : 2
             State : clean, degraded
   Working Devices : 1
    Failed Devices : 1
   ⋮
    Number  Major  Minor   RaidDevice State
       1      8      97        -      faulty /dev/sdf1
```

Чтобы заменить устройство `/dev/sdf1`, пометьте его как неисправное (если оно еще не является таковым) и удалите из RAID-массива:

```
→ sudo mdadm --manage /dev/md1 --fail /dev/sdf1
→ sudo mdadm --manage /dev/md1 --remove /dev/sdf1
```

Выключите компьютер, отсоедините кабель питания и замените вышедшее из строя устройство хранения новым такого же объема или больше. Я буду называть его вымышленным именем `/dev/NEW`, чтобы легко отличать в примерах, так как показанные далее команды могут нанести непоправимый вред компьютеру, а я этого не хочу. В реальных условиях укажите правильное имя устройства в системе.

Перезагрузите компьютер, определите новый диск в RAID-массиве (в нашем случае это `/dev/sdg`) и скопируйте его таблицу разделов на новое устройство командой `sgdisk`:

```
→ sudo sgdisk -R /dev/NEW /dev/sdg   Копирование данных из sdg в NEW
→ sudo sgdisk -G /dev/NEW            Генерация идентификаторов GUID
```

На устройстве `/dev/NEW` теперь есть раздел `/dev/NEW1` объемом 10 Гбайт. Добавьте его в массив:

```
→ sudo mdadm --manage /dev/md1 --add /dev/NEW1
mdadm: added /dev/NEW1
```

Массив немедленно начинает перестраиваться, зеркалируя данные на новое устройство:

```
→ cat /proc/mdstat
⋮
[==========>.........]  recovery = 51.5% ...
                       finish=0.4min ...
```

После завершения операции копирования данных новое устройство `/dev/NEW1` заменит неисправное устройство `/dev/sdf1`:

```
→ cat /proc/mdstat
Personalities : [raid1]
md1 : active raid1 NEW1[1] sdg1[0]
      10474496 blocks super 1.2 [2/2] [UU]
```

Удаление массива RAID

Если вы хотите удалить массив и использовать разделы для других целей, выполните следующие команды (предположим, что имена ваших устройств — `/dev/sdg1` и `/dev/sdh1`):

```
→ sudo umount /mnt/raid
→ sudo mdadm --stop /dev/md1
mdadm: stopped /dev/md1
→ sudo mdadm --zero-superblock /dev/sdg1 /dev/sdh1
```

Удалите из `/etc/fstab` и `/etc/mdadm/mdadm.conf` RAID-массив `/dev/md1` и сообщите об этом системе:

```
→ sudo update-initramfs –u
```

Хранение данных в системе LVM

`pvcreate`	Создание физического тома
`pvdisplay`	Просмотр информации о физических томах
`pvremove`	Удаление физического тома
`pvs`	Просмотр дополнительной информации о физических томах
`vgcreate`	Создание группы томов
`vgdisplay`	Просмотр информации о группе томов
`vgextend`	Добавление физического тома в группу томов
`vgreduce`	Удаление физического тома из группы томов
`vgremove`	Удаление группы томов
`lvcreate`	Создание логического тома
`lvdisplay`	Просмотр информации о логическом томе
`lvresize`	Изменение размера логического тома
`lvremove`	Удаление логического тома

Система управления логическими томами (LVM) решает две большие проблемы, связанные с хранением данных на дисках.

- **Ограниченный размер.** Когда диск заполнятся, приходится удалять данные или заменять сам диск новым.
- **Фиксированность разделов.** При разбивке диска на разделы вы решаете, сколько места займет каждый из них. Если вы ошибетесь, изменение займет много времени.

LVM решает эти проблемы путем создания уровня абстракции вокруг физического хранилища. Она совмещает физические диски любого размера, называемые *физическими томами*, в один большой диск, называемый *группой томов*. Группа томов становится рабочей площадкой для создания смоделированных разделов, называемых *логическими томами*, которые можно увеличивать и уменьшать. На рис. 4.1 показана взаимосвязь между физическими томами (PV), группой томов (VG), в которой они находятся, и логическими томами (LV), выделенными из общего пространства. Если в VG закончилось место, просто добавьте еще один физический том, и объем группы томов увеличится. Если LV имеет

неправильный размер, просто измените его. Существующие файлы сохраняются. Любые устройства хранения данных могут быть частью группы томов, даже RAID-массивы, созданные командой `mdadm` (см. раздел «RAID-массивы для резервирования» ранее в данной главе).

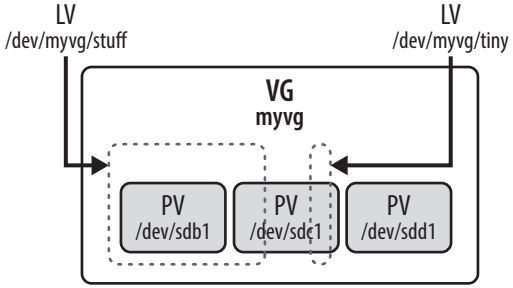


Рис. 4.1. Концепция LVM. Физические тома объединяются в VG. Логические тома располагаются вне VG

Самое популярное ПО для управления логическими томами в Linux называется `lvm2`. В него входит более 50 команд, и у них очень простые названия: сначала указывается тип — `pv`, `vg` или `lv`, а затем — глагол, например `create`, `remove` или `display`. Так, команда `vgcreate` создает группу томов, а `pvdisplay` выводит информацию о физическом томе.

ВНИМАНИЕ

Команды для управления логическими томами могут уничтожить вашу файловую систему без предупреждения. Для безопасности практикуйте команды на резервных дисках или на виртуальной машине.

К тому же концепция логических томов не очень надежна. Если один физический том выйдет из строя, вы потеряете всю группу томов. Для большей безопасности запустите LVM поверх RAID (см. раздел «RAID-массивы для резервирования» ранее).

Сейчас я приведу примеры использования наиболее распространенных команд с тремя пустыми 10-гигабайтными

дисковыми разделами: `/dev/sdb1`, `/dev/sdc1` и `/dev/sdd1`[1]. Помимо этих примеров, в `lvm2` доступны еще десятки команд. Для получения полного списка команд просмотрите документацию (в конце) к любой команде `lvm2` или посетите сайт sourceware.org/lvm2 (https://oreil.ly/yLLli).

Создание первого логического тома

В рамках этого шага мы установим два физических тома, объединим их в группу томов `myvg` на 20 Гбайт и создадим логический том `stuff` на 15 Гбайт:

```
→ sudo pvcreate /dev/sdb1 /dev/sdc1   Создание двух физических томов
  Physical volume "/dev/sdb1" successfully created.
  Physical volume "/dev/sdc1" successfully created.
→ sudo vgcreate myvg /dev/sdb1 /dev/sdc1   Создание группы томов
  Volume group "myvg" successfully created
→ sudo lvcreate -L 15G -n stuff myvg   Создание логического тома
  Logical volume "stuff" created.
→ sudo pvs                             Просмотр физических томов
  PV         VG   Fmt  Attr PSize    PFree
  /dev/sdb1  myvg lvm2 a--  <10.00g      0
  /dev/sdc1  myvg lvm2 a--  <10.00g  5.34g
```

Логический том `stuff` можно использовать так же, как и любое другое устройство хранения. Отформатируйте и смонтируйте его:

```
→ sudo mke2fs /dev/myvg/stuff       Форматирование логического тома
→ sudo mkdir /mnt/stuff             Монтирование тома
→ sudo mount /dev/myvg/stuff /mnt/stuff
→ df -h /mnt/stuff                            Просмотр тома
Filesystem             Size Used Avail Use% Mounted on
/dev/mapper/myvg-stuff  15G  24K  1.4G   1% /mnt/stuff
```

Подготовив логический том, добавьте в `/etc/fstab` следующую строку, чтобы монтировать его во время загрузки:

```
/dev/mapper/myvg-stuff  /mnt/stuff  ext4  defaults  0 2
```

---

[1] Управляйте разделами, а не целыми дисками (https://oreil.ly/Gl2AZ).

Просмотр информации о LVM

Команды `pvdisplay`, `vgdisplay` и `lvdisplay` выводят подробную информацию о физических томах, группах томов и логических томах соответственно. Команды `pvs`, `vgs` и `lvs` выводят полезные сводки этой информации:

```
→ sudo pvdisplay                Перечисление всех физических томов
→ sudo pvdisplay /dev/sdb1      Показать указанные физические тома
→ sudo pvs                      Вывод информации обо всех физических томах
→ sudo vgdisplay                Перечисление всех групп томов
→ sudo vgdisplay myvg           Показать указанные группы томов
→ sudo vgs                      Вывод информации обо всех группах томов
→ sudo lvdisplay                Перечисление всех логических томов
→ sudo lvdisplay myvg/stuff     Перечисление указанных логических томов
→ sudo lvs                      Вывод информации обо всех логических томах
```

Добавление логического тома

Снова запустим команду `lvcreate`, чтобы добавить в группу томов логический том `tiny` размером 2 Гбайта:

```
→ sudo lvcreate -L 2G -n tiny myvg
  Logical volume "tiny" created.
→ sudo mke2fs /dev/myvg/tiny                Форматирование
→ sudo mkdir /mnt/tiny                      Монтирование
→ sudo mount /dev/myvg/tiny /mnt/tiny
→ df -h /mnt/tiny                           Просмотр
Filesystem              Size Used Avail Use% Mounted on
/dev/mapper/myvg-tiny   2.0G   24K  1.9G   1% /mnt/tiny
```

Добавление диска в группу томов

Команда `vgextend` добавляет физические тома в группу томов. Предположим, что вы хотите увеличить размер диска на 10 Гбайт — до 25 Гбайт, но в группе томов `myvg` осталось только 3 Гбайта доступного пространства. Увеличьте общий объем группы томов до 30 Гбайт, добавив третий физический том, `/dev/sdd1`:

```
→ sudo pvcreate /dev/sdd1       Создание еще одного физического тома
→ sudo vgextend myvg /dev/sdd1  Увеличение размера группы томов
```

На этом этапе конфигурация LVM выглядит так, как показано на рис. 4.1. Запустите команду `lsblk`, чтобы вывести конфигурацию LVM:

```
→ lsblk /dev/sdb /dev/sdc /dev/sdd
NAME           MAJ:MIN RM SIZE RO TYPE MOUNTPOINTS
sdb              8:16   0  10G  0 disk
└─sdb1           8:17   0  10G  0 part
  └─myvg-stuff 253:0    0  15G  0 lvm  /mnt/stuff
sdc              8:32   0  10G  0 disk
└─sdc1           8:33   0  10G  0 part
  ├─myvg-stuff 253:0    0  15G  0 lvm  /mnt/stuff
  └─myvg-tiny  253:1    0  2G   0 lvm  /mnt/tiny
sdd              8:48   0  10G  0 disk
└─sdd1           8:49   0  10G  0 part
```

Увеличение логического тома

Команда `lvresize` увеличивает или уменьшает логический том[1]. Давайте увеличим объем логического тома до 25 Гбайт:

```
→ sudo lvresize --resizefs --size 25G /dev/myvg/stuff
→ df -h /mnt/stuff
Filesystem             Size Used Avail Use% Mounted on
/dev/mapper/myvg-stuff  25G  24K   24G   1% /mnt/stuff
```

Уменьшение логического тома

Оказывается, логический том не должен быть таким большим. Уменьшите его до 8 Гбайт, предварительно убедившись, что данными занято менее 8 Гбайт:

```
→ sudo lvresize --resizefs --size 8G /dev/myvg/stuff
Do you want to unmount "/mnt/stuff" ? [Y|n] y
⋮
Logical volume myvg/stuff successfully resized.
→ df -h /mnt/stuff
Filesystem             Size Used Avail Use% Mounted on
/dev/mapper/myvg-stuff  7.9G  24K  7.5G   1% /mnt/stuff
```

[1] Бывалые пользователи меняли размер логического тома, выполняя последовательно команды lvextend, umount, fsck, resize2fs и mount. Работать с командой lvresize куда проще.

Удаление логического тома

Команда `lvremove` удаляет логический том. Давайте избавимся от логического тома `tiny`:

```
→ sudo umount /mnt/tiny              Размонтирование логического тома
→ sudo lvremove /dev/myvg/tiny       Удаление логического тома
Do you really want to remove and DISCARD
active logical volume myvg/tiny? [y/n]: y
  Logical volume "tiny" successfully removed
```

Уменьшение группы томов

Команда `vgreduce` удаляет неиспользуемый физический том из группы томов. *Физический том* остается под управлением группы томов `lvm2`, но не в составе *группы томов* `myvg`:

```
→ sudo vgreduce myvg /dev/sdd1
  Removed "/dev/sdd1" from volume group "myvg"
```

Удаление группы томов

Команда `vgremove` удаляет группу томов, а также входящие в нее логические тома:

```
→ sudo vgremove myvg
Do you really want to remove volume group "myvg"
containing 1 logical volumes? [y/n]: y
Do you really want to remove and DISCARD
active logical volume myvg/stuff? [y/n]: y
  Logical volume "stuff" successfully removed
  Volume group "myvg" successfully removed
```

Удаление физического тома

Команда `pvremove` удаляет из LVM физические тома:

```
→ sudo pvremove /dev/sdb1 /dev/sdc1
  Labels on physical volume "/dev/sdb1" wiped.
  Labels on physical volume "/dev/sdc1" wiped.
```

ZFS — современная универсальная файловая система

`zpool`	Настройка пула хранения ZFS
`zfs`	Настройка набора данных ZFS

ВНИМАНИЕ!

Операции ZFS могут уничтожить файловую систему без подтверждения. Для безопасности пробуйте команды на резервных дисках или на виртуальных машинах.

ZFS (Zettabyte File System) объединяет в себе такие передовые функции, как RAID, LVM, шифрование и сжатие. Если вы привыкли использовать традиционные файловые системы Linux, например ext4, то ZFS покажется вам чем-то сверхъестественным. У нее своя терминология с «пулами» и «виртуальными устройствами». В ней не применяется путь `/etc/fstab` или команда `mount`. И вам даже не придется разбивать и форматировать диски.

Виртуальные устройства ZFS (сокращенно *vdev*) — это группа физических дисков, работающих вместе. Они могут распределять данные между собой, словно это один большой диск, например RAID-0. Диски могут зеркально копировать данные друг друга, например, в системе RAID-1. Они могут работать в качестве дискового кэша, доступно и множество других возможностей.

Совокупность виртуальных устройств называется *пулом*. Пул действует как одно большое устройство хранения. Вы можете разделить его на части — *наборы данных* и гибко изменять ограничения на их размер и другие атрибуты. Имена наборов данных выглядят как пути Linux без ведущего слеша. Например, пул с именем `mypool` и набором данных `stuff` будет называться `mypool/stuff`. (Наборы данных могут быть вложенными, например `mypool/stuff/important`.) Добавьте ведущий слеш, и у вас получится точка монтирования набора данных по умолчанию в Linux, например `/mypool/stuff`.

ZFS — не единственная файловая система с расширенными возможностями, популярна также Btrfs, но ZFS проще в настройке.

ПРИМЕЧАНИЕ

Я рассматриваю не все возможности ZFS. Настоящие ZFS требуют тщательного выстраивания конфигурации, настройки и большого количества оперативной памяти (читайте документацию, перейдя по ссылке https://oreil.ly/-t5Fu).

Чтобы продемонстрировать работу ZFS с созданием зеркал и чередованием, я использую две пары дисков (рис. 4.2). Каждая пара представляет собой виртуальное устройство для зеркалирования данных (RAID-1). Затем ZFS чередует эти два виртуальных устройства (RAID-0), создавая RAID-10. Этот пул может выдержать отказ диска в каждом виртуальном устройстве и сохранить данные.

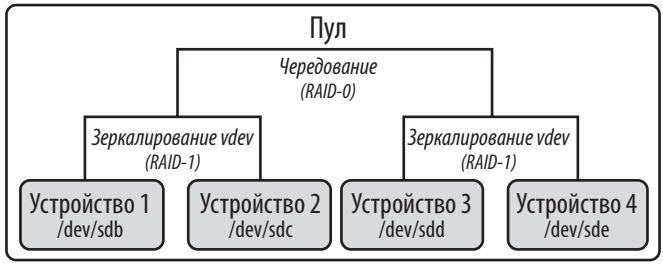


Рис. 4.2. Мой пример конфигурации ZFS — RAID-10

Создание пула ZFS

С помощью команды `zpool` создадим пул из двух пар дисков для зеркалирования данных. Я создал пул `mypool` из четырех дисковых устройств емкостью 10 Гбайт, `/dev/sdb`, `/dev/sdc`, `/dev/sdd` и `/dev/sde`:

```
→ sudo zpool create mypool \
  mirror /dev/sdb /dev/sdc \
  mirror /dev/sdd /dev/sde
```

ВНИМАНИЕ!

Простые имена устройств, например /dev/sdb, после перезагрузки могут измениться. Для создания более надежной конфигурации используйте имена, которые точно не изменятся, например символические ссылки в /dev/disk/by-id или /dev/disk/by-uuid.

В реальных системах при создании пула не забудьте задать соответствующее значение двоичного сдвига с помощью команды -o ashift (см. документацию).

Для просмотра результатов используйте команду `zpool status`:

```
→ zpool status
  pool: mypool
 state: ONLINE
config:
  NAME          STATE     READ   WRITE   CKSUM
  mypool        ONLINE       0       0       0   Пул
    mirror-0    ONLINE       0       0       0   Первое виртуальное
                                                  устройство
      sdb       ONLINE       0       0       0
      sdc       ONLINE       0       0       0
    mirror-1    ONLINE       0       0       0   Второе виртуальное
                                                  устройство
      sdd       ONLINE       0       0       0
      sde       ONLINE       0       0       0
```

Создание набора данных ZFS

В традиционных файловых системах используются разделы фиксированного размера, которые вы монтируете в файле `/etc/fstab`. В ZFS задействуются наборы данных произвольного размера, которые монтируются автоматически. Создайте набор данных `data` в пуле `mypool`, смонтированном в каталоге `/mypool/data`:

```
→ sudo zfs create -o mypool/data
```

Переместите точку монтирования в `/mnt/stuff`:

```
→ sudo zfs set mountpoint=/mnt/stuff mypool/data
```

Посмотрите результаты с помощью любой из перечисленных далее команд:

```
→ zfs mount
mypool            /mypool                    Весь пул
mypool/data       /mnt/stuff                 Ваш набор данных
→ zfs get mountpoint mypool/data
NAME          PROPERTY    VALUE       SOURCE
mypool/data   mountpoint  /mnt/stuff  local
```

Теперь используйте набор данных как любой другой смонтированный раздел:

```
→ sudo cp /etc/hosts /mnt/stuff
→ cd /mnt/stuff
→ ls
hosts
```

Создание зашифрованного набора данных ZFS

Добавив пару параметров, вы можете создать зашифрованный набор данных. Перед тем, как изменить его, нужно будет ввести пароль. Создадим зашифрованный набор данных `mypool/cryptic`:

```
→ zfs create \
  -o encryption=on \
  -o keylocation=prompt \
  -o keyformat=passphrase \
  mypool/cryptic
Enter new passphrase: xxxxxxxx
Re-enter new passphrase: xxxxxxxx
```

Используйте набор данных в обычном режиме. Когда вы перезагрузитесь или нужно будет изменить его, выполните команду

```
→ sudo zfs mount -l mypool/cryptic
```

Ограничение размера наборов данных ZFS

По умолчанию набор данных ZFS имеет тот же размер, что и пул. Вы можете ограничить размер набора указанием квоты, которую можно изменить в любой момент:

```
→ sudo zfs set quota=15g mypool/data
→ zfs list
NAME          USED  AVAIL REFER  MOUNTPOINT
mypool        312K  18.4G   25K  /mypool
mypool/data    24K  15.0G   24K  /mnt/stuff      Лимит 15 Гбайт
```

Сжатие наборов данных ZFS

ZFS может автоматически сжимать данные при записи и распаковывать их при чтении. Поддерживаются различные алгоритмы сжатия, я использую сжатие `gzip` и просматриваю, насколько эффективно сжимаются файлы (коэффициент сжатия):

```
→ sudo zfs set compression=gzip mypool/data
→ cp hugefile /mnt/stuff                    Сохранение большого файла
→ zfs get compressratio                     Просмотр степени сжатия
NAME           PROPERTY        VALUE  SOURCE
mypool         compressratio   122.66x  -
mypool/data    compressratio   126.12x  -
```

После включения сжатия сжиматься будут только новые данные, а старые останутся неизменными. Чтобы отключить сжатие, выполните команду

```
→ sudo zfs set compression=off mypool/data
```

Создание моментального снимка набора данных ZFS

ZFS поддерживает *моментальные снимки* — сохранение состояния набора данных, к которому впоследствии можно легко вернуться (откатиться). Например, перед выполнением рискованного изменения сделайте снимок файлов, и если что-то пойдет не так, вы вернетесь в исходное состояние с помощью одной команды. Снимки не занимают много места на диске, и вы можете эффективно отправлять их в другие пулы или хосты с помощью команд `zfs send` и `zfs recv`. Создайте снимок `mypool/data` с именем `safe`:

```
→ sudo zfs snapshot mypool/data@safe
```

Отобразите список снимков:

```
→ zfs list -t snapshot
NAME               USED   AVAIL     REFER  MOUNTPOINT
mypool/data@safe     0B       -       24K  -
```

Если список снимков не отображается, настройте свойство `listsnapshots=on` и повторите попытку:

```
→ sudo zpool set listsnapshots=on mypool
```

Попробуйте изменить некоторые файлы в `mypool/data`. Затем откатитесь к начальному состоянию и проверьте, что все изменения исчезли:

```
→ sudo zfs rollback mypool/data@safe
```

Уничтожение набора данных или моментального снимка ZFS

Будьте осторожны с командой `zfs destroy` — она выполняется мгновенно и не требует подтверждения:

```
→ sudo zfs destroy mypool/data@safe      Моментальный снимок
→ sudo zfs destroy mypool/data           Набор данных
```

Удаление пула ZFS

Будьте осторожны с командой `zpool destroy` — она выполняется мгновенно и не требует подтверждения:

```
→ sudo zpool destroy mypool
```

Резервное хранение и удаленное хранение

`rsync`	Эффективное копирование файлов даже по Сети
`rclone`	Синхронизация файлов с разными облачными провайдерами
`dd`	Низкоуровневое копирование данных
`growisofs`	Запись DVD и Blu-ray-дисков

Вы можете создавать резервные копии ценных файлов Linux разными способами:

- копировать их на удаленную машину;
- копировать на резервный носитель, например внешний диск;
- записывать на оптический диск.

Я привожу несколько популярных команд Linux, но их намного больше. Некоторые пользователи любят `cpio` за ее гибкость, а другие уверены, что `dump` и `restore` — это единственный надежный способ резервного копирования и восстановления файлов любого типа. Если вас заинтересовали эти команды, обратитесь к документации.

---

rsync

stdin **stdout** -file **--opt** **--help** **--version**

`rsync [`*`параметр(ы)`*`] `*`источник назначение`*

Команда `rsync` копирует набор файлов. Она может создать точную копию, включая права доступа и другие атрибуты файла, то есть выполнить так называемое *зеркалирование*, а также скопировать сами данные. Команда может работать по сети или локально. `rsync` имеет множество применений и более 50 вариантов, я приведу лишь несколько распространенных случаев, связанных с резервным копированием.

Чтобы локально отзеркалить каталог `mydir` и его содержимое в каталог `mydir2`, выполните команду

```
→ rsync -a mydir mydir2
```

Команда `rsync` весьма придирчиво относится к тому, как вы указываете первый каталог. Если вы пишете `mydir`, как в приведенном примере, то он будет скопирован в `mydir2` с созданием подкаталога `mydir2/mydir`. Если вы хотите, чтобы в `mydir2` копировалось только содержимое каталога `mydir`, добавьте к `mydir` слеш:

```
→ rsync -a mydir/ mydir2
```

Команда `rsync` может создать зеркало каталога по сети на другом хосте, защитив соединение с помощью протокола SSH для предотвращения перехвата данных. В примере

далее я копирую каталог `mydir` в учетную запись `smith` на удаленном хосте `server.example.com`, в каталог `D2`:

```
→ rsync -a mydir smith@server.example.com:D2
```

Если вам пришлась по душе команда `rsync`, но вы хотите также создавать резервные копии и управлять ими, попробуйте `rsnapshot` (https://oreil.ly/VNfb-).

Полезные параметры

`-o` Копирование владельцев файлов (возможно, вам понадобятся права суперпользователя на удаленном хосте)

`-g` Копирование принадлежности к группам файла (возможно, вам понадобятся права суперпользователя на удаленном хосте)

`-p` Копирование прав доступа к файлу

`-t` Копирование меток времени файла

`-r` Рекурсивное копирование каталогов (то есть включая их содержимое)

`-l` Копирование символических ссылок (а не файлов, на которое они указывают)

`-D` Копирование устройств (только для суперпользователя)

`-a` Зеркалирование — копирование всех атрибутов исходных файлов. Подразумевает все параметры `-Dogptrl`

`-x` При копировании дерева файлов оставаться в пределах текущей файловой системы, не переходить в другие

`-z` Сжатие данных для передачи. Полезно только для удаленных хостов с медленным соединением. Избегайте сжатия при локальном копировании файлов

`-n` Режим формального прогона — без фактического копирования. Вывод *предполагаемых* задач

`-v` Режим расширенного вывода — вывод информации о состоянии во время копирования. Чтобы отображать числовой индикатор выполнения копирования файлов, добавьте `--progress`

---

rclone

stdin stdout -file --opt --help --version

`rclone` *команда* [*параметр(ы)*] [*аргумент(ы)*]

Команда `rclone` подключает Linux-систему к популярным облачным хранилищам для удобного копирования файлов. Она поддерживает службы Dropbox, Google Drive, Microsoft OneDrive, Amazon S3 и еще около 50 репозиториев. Чтобы

начать работу, запустите команду `rclone config` и установите соединение с выбранным облачным провайдером, которого команда `rclone` называет *удаленным*. Подробные инструкции по настройке можно найти на сайте rclone.org/docs (https://oreil.ly/YXVrw).

Выбрав подходящее имя, скажем, `myremote`, вы можете обращаться к удаленным файлам с помощью синтаксиса `myremote:`*`путь`*, указав *`путь`* к файлу. Например, файл `photo.jpg` в удаленном каталоге `Photos` будет иметь имя `myremote:Photos/photo.jpg`.

Обычно резервное копирование выполняется с помощью команды `rclone sync`, которая синхронизирует локальный и удаленный каталоги. Команда автоматически добавляет или удаляет содержимое каталогов, синхронизируя их. Выполняется так же, как и команда `rsync --delete`. Синхронизацию можно реализовать в любом порядке: с локальной машины на удаленную или с удаленной на локальную. Можно даже настроить шифрование на стороне клиента, чтобы файлы были зашифрованы перед копированием на удаленную машину и дешифрованы при копировании обратно в локальную систему (см. документацию).

Перечислю некоторые операции для резервного копирования.

`rclone ls remore:`	Рекурсивное перечисление всех файлов на удаленной машине (чтобы перечислить без рекурсии, добавьте `--max-depth 1`)
`rclone lsd remote:`	Перечисление каталогов только на удаленной машине
`rclone lsl remote:`	Рекурсивное перечисление всех файлов на удаленной машине в виде длинного списка (как `ls -l`) (без рекурсии — добавьте `--max-depth 1`)
`rclone copy` *`файл`* `remote:`	Копирование локального *файла* на удаленную машину
`rclone copy remote:`*`файл`* `.`	Копирование удаленного *файла* в локальную систему
`rclone move` *`файл`* `remote:`	Перемещение локального *файла* на удаленную машину
`rclone move remote:`*`файл`* `.`	Перемещение удаленного *файла* в локальную систему
`rclone delete remote:`*`файл`*	Удаление *файла* на удаленной машине

`rclone sync` *`каталог remote: каталог`*	Синхронизация локальных файлов на удаленной машине
`rclone sync remote:` *`каталог каталог`*	Синхронизация удаленных файлов в локальной системе

Выполните команду `rclone help` для получения полного списка подкоманд:

```
→ rclone copy remote:Photos/Vacation/picture.jpg .
```

Обратите внимание на то, что на удаленном компьютере можно получить доступ к более сложным путям, чем показано в моих примерах.

Полезные параметры

`-n`	Режим формального прогона, без фактического копирования. Вывод *предполагаемых* задач
`-i`	Интерактивный режим с подтверждением каждого изменения вручную

---

dd

stdin **stdout** -file --opt **--help** **--version**

`dd [`*`параметр(ы)`*`]`

`dd` — это инструмент низкоуровневого побайтового копирования данных. Позволяет копировать данные из одного файла в другой, например из `myfile` в `/tmp/mycopy`:

```
→ dd if=myfile of=/tmp/mycopy
2+1 records in
2+1 records out
1168 bytes (1.2 kB) copied, 0.000174074 s, 6.7 MB/s
```

Команда позволяет преобразовывать данные во время копирования, например менять символы на прописные при копировании из `myfile` в `/tmp/mycopy`:

```
→ dd if=myfile of=/tmp/mycopy conv=ucase
```

Команда `dd` способна на гораздо большее, чем простое копирование файлов. Она может копировать необработанные (raw) данные непосредственно с одного дискового

устройства на другое. Далее показана команда для клонирования жесткого диска:

```
→ sudo dd if=/dev/device1 of=/dev/device2 bs=512 \
  conv=noerror,sync                    Перезапись /dev/device2
```

Можно также скопировать все дисковое устройство для создания файла ISO. Убедитесь, что выходной файл находится на другом дисковом устройстве и имеет достаточно свободного пространства:

```
→ sudo dd if=/dev/device of=disk_backup.iso
```

ВНИМАНИЕ!

Команда dd, запущенная от имени суперпользователя, может уничтожить данные на жестком диске за несколько секунд — будьте осторожны. Всегда проверяйте, что выходной файл (аргумент of=) — тот самый, который вы хотите получить. Прежде чем экспериментировать с командой dd от имени суперпользователя, создайте резервную копию компьютера и держите при себе live-диск с Linux (см. раздел «О чем эта книга» в самом начале).

Отличные советы по применению команды dd приведены на сайте https://oreil.ly/R3krx. Мой любимый совет — создание резервной копии главной загрузочной записи (MBR) диска длиной 512 байт в файле `mbr.txt`:

```
→ sudo dd if=/dev/device of=mbr.txt bs=512 count=1
```

Полезные параметры

`if=`*файл*	Использование указанного *файла* или устройства в качестве источника
`of=`*файл*	Использование указанного *файла* или устройства в качестве объекта назначения. **Внимательно проверьте правильность своего выбора!**
`bs=`*N*	Копирование *N* байт (размер блока) за раз. (Чтобы задать разные параметры блока источника и объекта назначения, используйте `ibs` и `obs` соответственно)
`skip=`*N*	Пропуск *N* блоков источника (`ibs`) перед началом копирования
`seek=`*N*	Без учета *N* блоков объекта назначения (`obs`) перед началом копирования

conv=*операция* Преобразование копируемых данных. *Операцией* может быть `ucase` (преобразовать в прописные), `lcase` (преобразовать в строчные), `ascii` (преобразовать в ASCII из EBCDIC) и т. д. Больше информации вы найдете в документации

growisofs

stdin stdout **-file** **--opt** **--help** **--version**

```
growisofs [параметр(ы)] дорожки
```

Команда `growisofs` позволяет записать данные на CD, DVD или Blue-Ray-диски. Чтобы записать содержимое каталога Linux на диск, читаемый в системах Linux, Windows и macOS, сделайте следующие шаги.

1. Найдите устройство записи дисков, выполнив команду

   ```
   → grep "^drive name:" /proc/sys/dev/cdrom/info
   drive name:         sr1      sr0
   ```

   В примере доступны следующие устройства: `/dev/sr1` и `/dev/sr0`.

2. Поместите файлы для записи в каталог `dir`. Расположите их в нужном порядке. Сам каталог `dir` на диск не копируется — только его содержимое.

3. Используйте команду `mkisofs` для создания файла образа ISO (диска) и запишите его на диск с помощью команды `growisofs` (в примере задействуется устройство `/dev/sr1`):

   ```
   → mkisofs -R -l -o $HOME/mydisk.iso dir
   → growisofs -dvd-compat -Z /dev/sr1=$HOME/mydisk.iso
   → rm $HOME/mydisk.iso
   ```

Для записи аудио-CD используйте другую подходящую программу с графическим интерфейсом, например `k3b`.

ГЛАВА 5

Сетевые команды

Информация о хосте

`uname`	Вывод основной информации о системе
`hostname`	Вывод имени хоста системы
`ip`	Вывод информации о сетевом интерфейсе и управление ею

У каждой машины Linux есть сетевое имя, IP-адрес и другие характеристики. Сейчас я покажу, как отобразить эту информацию.

uname

stdin **stdout** -file **--opt** **--help** **--version**

```
uname [параметр(ы)]
```

Команда `uname` выводит информацию об операционной системе, в том числе о ядре:

```
→ uname -a
Linux myhost 5.15.0-76-generic #83-Ubuntu
 SMP Thu Jun 15 19:16:32 UTC 2023 x86_64
 x86_64 x86_64 GNU/Linux
```

В выводе указывается название ядра (`Linux`), имя хоста (`myhost`), выпуск ядра (`5.15.0-76-generic`), версия ядра (`#83-Ubuntu SMP Thu Jun 15 19:16:32 UTC 2023`), тип оборудования (`x86_64`), тип процессора (также `x86_64`), аппаратная платформа (также `x86_64`) и название ОС (`GNU/Linux`).

Полезные параметры

`-a` Вывод полной информации
`-s` Вывод только названия ядра (по умолчанию)
`-n` Вывод только имени хоста
`-r` Вывод только выпуска ядра
`-v` Вывод только версии ядра
`-m` Вывод только типа оборудования
`-p` Вывод только типа процессора
`-i` Вывод только аппаратной платформы
`-o` Вывод только названия ОС

hostname

stdin **stdout** -file **--opt** **--help** **--version**

`hostname [`*`параметр(ы)`*`] [`*`имя`*`]`

Команда `hostname` выводит имя вашего компьютера. В зависимости от настроек в выводе может быть указано полное имя хоста:

```
→ hostname
myhost.example.com
```

или короткое:

```
→ hostname
myhost
```

Вы также можете задать имя хоста при запуске команды от имени суперпользователя:

```
→ sudo hostname orange
```

ПРИМЕЧАНИЕ

Изменения, внесенные командой hostname, могут сбрасываться после перезагрузки. Прочитайте документацию к вашему дистрибутиву для получения дополнительной информации. Возможно, как вариант вам понадобится отредактировать файл конфигурации, который считывается во время загрузки.

Полезные параметры

`-s`	Вывод короткого имени вашего хоста
`-f`	Вывод полного имени вашего хоста
`-d`	Вывод DNS-имени домена вашего хоста
`-a`	Вывод всех псевдонимов вашего хоста
`-I`	Вывод всех сетевых адресов вашего хоста
`-F` *файл*	Задание имени хоста путем извлечения из указанного *файла*

---

ip

stdin **stdout** -file --opt **--help** --version

`ip [`*параметр(ы)*`] `*объект команда*

Команда `ip` отображает и настраивает различные параметры сетевого интерфейса вашего компьютера. Данная тема выходит за рамки тематики данной книги, тем не менее я познакомлю вас с несколькими командами.

Если вы знаете стандартное имя интерфейса, например `eth0`, `ens160` или `enp68s0`, то можете просмотреть подробную информацию о нем с помощью следующей команды:

```
→ ip addr show eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> ...
  link/ether 00:50:ba:48:4f:ba brd ff:ff:ff: ...
  inet 192.168.0.21/24 brd 192.168.0.255 scope ...
  inet6 fe80::21e:8cff:fe53:41e4/64 ...
```

В выводе указаны IPv4-адрес (`192.168.0.21`), IPv6-адрес (`fe80::21e:8cff:fe53:41e4/64`), MAC-адрес (`00:50:ba:48:4f:ba`) и прочие сведения. Если вы не знаете имени интерфейса, то можете отобразить все подключенные сетевые интерфейсы:

```
→ ip addr show | less
```

Другая команда, `ifconfig`, выдает информацию, аналогичную предоставляемой `ip addr show`:

```
→ ifconfig eth0          Информация об одном сетевом интерфейсе
→ ifconfig -a            Информация обо всех сетевых интерфейсах
```

Она считается в целом устаревшей, но вы все еще встретите ее во многих системах.

С помощью показанных далее команд `ip` выводится информация о сети. Добавьте параметр `help` в конец команды (например, `ip link help`), чтобы получить подробную информацию о коэффициенте нагрузки.

`ip addr`	Вывод IP-адресов сетевых устройств
`ip maddr`	Вывод адресов многоадресного вещания сетевых устройств
`ip link`	Отображение атрибутов сетевых устройств
`ip route`	Отображение таблицы маршрутизации
`ip monitor`	Запуск мониторинга сетевых устройств. Чтобы остановить мониторинг, нажмите ^C
`ip help`	Просмотр информации о правилах использования всех перечисленных ранее команд

Суперпользователи могут также применять команду `ip` для настройки сетевых устройств, управления таблицами и правилами маршрутизации, создания туннелей и т. д. Подробнее о команде `ip` и связанном с ней инструменте `iproute2` (https://oreil.ly/jT5FF) говорится в документации к `ip`.

Локация хоста

`host`	Поиск имен хостов, IP-адресов и информации DNS
`whois`	Поиск регистраторов интернет-доменов
`ping`	Проверка доступности удаленного хоста
`traceroute`	Просмотр сетевого пути к удаленному хосту

В процессе работы с удаленными компьютерами вам непременно захочется получить больше информации о них. Их владельцы? IP-адреса? Где они расположены?

host

stdin **stdout** -file **--opt** --help --version

```
host [параметр(ы)] имя [сервер]
```

Команда `host` определяет имя хоста или IP-адрес удаленной машины, запрашивая DNS:

```
→ host www.ubuntu.org
www.ubuntu.com has address 69.16.230.226
→ host 69.16.230.226
226.230.16.69.in-addr.arpa domain name pointer
 lb05.parklogic.com.
```

Команда может извлечь намного больше информации:

```
→ host -a www.ubuntu.org
Trying "www.ubuntu.org"
;; ->>HEADER<<- opcode: QUERY, status: NOERROR ...
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, ...

;; QUESTION SECTION:
;www.ubuntu.org.                  IN       ANY

;; ANSWER SECTION:
www.ubuntu.org.      60     IN    CNAME    ubuntu.org.
```

Однако разбор этого вывода выходит за рамки тематики данной книги. Последний, необязательный параметр *сервер* позволяет указать конкретный сервер имен для запроса:

```
→ host www.ubuntu.org ns1.parklogic.com
Using domain server:
Name: ns1.parklogic.com
Address: 69.16.230.48#53
Aliases:
www.ubuntu.org has address 69.16.230.226
www.ubuntu.org mail is handled by ...
```

Полезные параметры

Чтобы просмотреть все параметры, запустите команду `host` без них.

`-a` Вывод всей доступной информации о хосте

`-t` Тип запроса сервера имен: `A`, `AXFR`, `CNAME`, `HINFO`, `KEY`, `MX`, `NS`, `PTR`, `SIG`, `SOA` и т. д.

Далее приведен пример использования параметра `-t` для поиска записей MX:

```
→ host -t MX centos.org
centos.org mail is handled by 10 mail.centos.org.
```

Если команда `host` не удовлетворяет вашим потребностям, то используйте `dig` — другую утилиту поиска DNS или команду `nslookup`, которая устарела, но все еще поддерживается.

whois

stdin **stdout** -file **--opt** **--help** **--version**

```
whois [параметр(ы)] домен
```

Команда `whois` выводит информацию о регистрации интернет-домена. Вывод получается длинным, так что рекомендуется получать его с помощью команды `less`:

```
→ whois linuxmint.com | less
  Domain name: LINUXMINT.COM
  Registrar: Ascio Technologies, Inc. Danmark
  Updated Date: 2023-05-16T22:22:38Z
  Creation Date: 2006-06-07T10:45:34Z
  Name Server: NS01.SERVAGE.NET
  ⋮
```

Полезные параметры

`-h` *сервер* — Поиск на указанном *сервере* whois регистратора, например:

```
→ whois -h whois.comlaude.com redhat.com
```

`-p` *порт* — Опрос указанного TCP-*порта* вместо стандартного (43)

ping

stdin **stdout** -file **--opt** --help --version

```
ping [параметр(ы)] хост
```

Команда `ping` позволяет узнать, доступен ли удаленный *хост*. Она отправляет на него небольшие сообщения, называемые ICMP-пакетами, ожидает ответа и выводит информацию о том, через какое время пришел ответ:

```
→ ping slackware.com
PING slackware.com (64.57.102.36) 56(84) bytes of data.
64 bytes from connie.slackware.com (64.57.102.36):
 icmp_seq=1 ttl=52 time=107 ms
```

```
64 bytes from connie.slackware.com (64.57.102.36):
 icmp_seq=2 ttl=52 time=107 ms
^C
--- slackware.com ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, ...
rtt min/avg/max/mdev = 107.207/107.350/107.494/0.143 ms
```

В примере хост `slackware.com` ответил на пинг через 107 миллисекунд. (Обратите внимание на то, что его настоящее имя — `connie.slackware.com`.) Пинг может блокироваться брандмауэрами, поэтому и работающий хост может не ответить.

Полезные параметры

`-c` *N*	Пинг не более *N* раз
`-i` *N*	Ожидание между пингами *N* секунд (по умолчанию 1)
`-n`	Вывод IP-адресов вместо имен хостов
`-6`	Использование по умолчанию протокола IPv6 вместо IPv4

traceroute

stdin **stdout** -file **--opt** **--help** **--version**

```
traceroute [параметр(ы)] хост [размер_пакета]
```

Команда `traceroute` выводит сетевой путь от вашего локального узла до удаленного узла, а также затрачиваемое на это время:

```
→ traceroute archlinux.org
 1 server.mydomain.com (192.168.0.20) 1.397 ms ...
 2  10.221.16.1 (10.221.16.1) 15.397 ms ...
 3  router.example.com (92.242.140.21) 4.952 ms ...
 ⋮
12  archlinux.org (95.217.163.246)  117.100 ms ...
```

Команда посылает три «сигнала» на каждый хост на пути и сообщает время возврата. Если хост не отвечает 5 секунд, то `traceroute` печатает звездочку. Если `traceroute` блокируется брандмауэром или не может продолжить работу, то команда печатает символ из перечисленных в таблице.

Символ	Значение
!F	Необходима фрагментация
!H	Хост недоступен
!N	Сеть недоступна
!P	Протокол недоступен
!S	Маршрутизация от источника не удалась
!X	Связь запрещена администратором
!*N*	ICMP-код недоступности *N*

По умолчанию *размер_пакета* составляет 40 байт. Вы можете передать собственное значение (например, `traceroute myhost 120`). Для более интерактивной работы попробуйте команду `mtr` (*my traceroute*).

Полезные параметры

`-n` Числовой режим — использование IP-адресов вместо имен хостов

`-w N` Применение указанного значения тайм-аута от 5 секунд до *N* секунд

`-6` Использование по умолчанию протокола IPv6 вместо IPv4

Сетевые соединения

`ssh` Аутентификация на удаленном хосте и выполнение на нем команд

`scp` Копирование файлов между двумя хостами

`sftp` Интерактивное копирование файлов между двумя хостами

`netcat` Создание пользовательского сетевого соединения

В Linux можно легко и безопасно подключаться с одного узла на другой для аутентификации в удаленной системе, передачи файлов и других целей.

ssh

stdin **stdout** -file --opt --help --version

```
ssh [параметр(ы)] хост [команда]
```

Команда `ssh` обеспечивает безопасную аутентификацию на удаленной машине. Если ваши локальное и сетевое имена

пользователя идентичны, то просто укажите имя удаленного *хоста*:

```
→ ssh remote.example.com
smith@remote.example.com's password: xxxxxxxx
```

В противном случае укажите имя пользователя удаленного компьютера:

```
→ ssh sandy@remote.example.com          Вариант синтаксиса
→ ssh -l sandy remote.example.com       Другой вариант синтаксиса
```

Если вы передадите `ssh` другую *команду*, то `ssh` вызовет ее на удаленной машине без инициирования интерактивного входа в систему:

```
→ ssh sandy@remote.example.com df       Проверка свободного места на диске
```

Команда `ssh` шифрует все данные, передаваемые через сеть, включая ваше имя пользователя и пароль. Протокол SSH поддерживает и другие способы аутентификации, например, открытые ключи (см. врезку «Аутентификация по открытому ключу в SSH» далее).

Полезные параметры

`-l` *пользователь*	Использование сетевого имени *пользователя*, в противном случае `ssh` задействует его локальное имя. Или воспользуйтесь синтаксисом *пользователь@хост*: `→ ssh sandy@remote.example.com`
`-p` *порт*	Подключение к *порту*, отличному от используемого по умолчанию (22)
`-t`	Активация `tty` на удаленной машине. Полезна при попытке запустить сетевую команду с интерактивным пользовательским интерфейсом, например текстовым редактором
`-v`	Вывод подробного сообщения, полезного в целях отладки

Аутентификация по открытому ключу в SSH

Аутентификация по открытому ключу — это более безопасный способ авторизации на удаленных узлах по протоколу SSH. В этом случае используется пара криптографических ключей, которые хранятся на локальном хосте в зашифрованном каталоге `~/.ssh`. Ваш открытый ключ будет скопирован на удаленные узлы. Сначала сгенерируйте пару ключей:

```
→ ssh-keygen
Enter file to save the key (~/.ssh/id_rsa): <Enter>
Enter passphrase (empty for no passphrase): xxxxxxx
Enter same passphrase again:  xxxxxxx
Your identification has been saved in ~/.ssh/id_rsa
```

Затем скопируйте свой открытый ключ на удаленный хост и войдите в систему:

```
→ ssh-copy-id sandy@remote.example.com
sandy@remote.example.com's password: xxxxxxxx
Number of key(s) added: 1
→ ssh sandy@remote.example.com
Enter passphrase for '/home/sandy/.ssh/id_rsa': xxxxxxx
```

Если последняя команда `ssh` не помогла вам войти в систему, то проверьте следующее:

- у локального и удаленного каталогов `~/.ssh` права `0700`;
- у файла локального закрытого ключа `~/.ssh/id_rsa` права `0600`;
- у удаленного файла `~/.ssh/authorized_keys` права `600`;
- открытый ключ из локального файла `~/.ssh/id_rsa.pub` указан в удаленном файле `~/.ssh/authorized_keys`. Если его там нет, то добавьте содержимое локального файла в удаленный файл `authorized_keys` и повторите попытку.

Вы также можете выполнить команду `ssh -v` для вывода отладочных сообщений и подсказок. И команду `ssh -vv` для получения подробной информации.

scp

stdin **stdout** -file --opt --help --version

`scp` *исходные_данные целевые_данные*

Команда `scp` (*secure copy*) копирует файлы и каталоги с одного компьютера на другой в пакетном режиме. (Для запуска интерактивного пользовательского интерфейса обратитесь к команде `sftp`.) Она шифрует все соединения между двумя машинами с помощью протокола SSH. Приведу простой пример — команда `scp` копирует локальный файл на удаленную машину:

```
→ scp исходный_файл remote.example.com:копия_файла
```

Можно рекурсивно скопировать каталог на удаленную машину:

```
→ scp -r исходный_каталог remote.example.com:
```

Можно скопировать удаленный файл на локальную машину:

```
→ scp remote.example.com:файл .
```

Или можно рекурсивно скопировать удаленный каталог на локальную машину:

```
→ scp -r remote.example.com:каталог .
```

Вы даже можете скопировать файлы с одной удаленной машины на другую, если у вас есть SSH-доступ к обеим:

```
→ scp remote1.example.com:файл remote2.example.com:
```

Чтобы применить альтернативное имя пользователя на удаленной системе, вооружитесь синтаксисом *пользователь@хост*:

```
→ scp файл sandy@remote.example.com:
```

Полезные параметры

`-P` *порт*	Подключение к указанному TCP-*порту* (по умолчанию 22)
`-p`	Сохранение при копировании всех атрибутов файла (прав доступа, меток времени)
`-r`	Рекурсивное копирование каталогов и их содержимого
`-v`	Полный вывод, полезный для отладки

sftp

stdin stdout -file **--opt --help --version**

```
sftp (хост | пользователь@хост)
```

Команда `sftp` — это интерактивная программа, которая безопасно копирует файлы между двумя компьютерами с помощью SSH (избегайте похожей команды `ftp` — в большинстве случаев она передает имена пользователей, пароли и файлы по сети в незащищенном виде):

```
→ sftp remote.example.com
smith@remote.example.com's password: xxxxxxxx
sftp> cd myfiles                    Изменение удаленного каталога
sftp> ls                            Перечисление удаленных каталогов
README
file1
file2
file3
sftp> get file2                     Передача удаленного файла
Fetching /home/sandy/MyFiles/file2 to file2
sftp> quit
```

Если ваше имя пользователя на удаленной системе отличается от локального, задействуйте синтаксис *пользователь@хост*:

```
→ sftp sandy@remote.example.com
```

Команда	Значение
`help`	Вывод списка доступных команд
`ls`	Вывод списка файлов в текущем удаленном каталоге
`lls`	Вывод списка файлов в текущем локальном каталоге
`pwd`	Вывод удаленного рабочего каталога
`lpwd`	Вывод локального рабочего каталога
`cd` *каталог*	Смена удаленного каталога на указанный
`lcd` *каталог*	Смена локального каталога на указанный
`get` *файл1* [*файл2*]	Копирование удаленного *файла1* на локальную машину, при необходимости можно переименовать его в *файл2*
`put` *файл1* [*файл2*]	Копирование локального *файла1* на удаленную машину, при необходимости можно переименовать его в *файл2*

Команда	Значение
`mget file*`	Копирование нескольких удаленных файлов на локальную машину с помощью шаблонов * и ?
`mput file*`	Копирование нескольких локальных файлов на удаленную машину с помощью шаблонов * и ?
`quit`	Выход из `sftp`

netcat

stdin **stdout** -file --opt --help --version

```
netcat [параметр(ы)] [назначение] [порт]
nc [параметр(ы)] [назначение] [порт]
```

`netcat` (`nc`) — это универсальный инструмент для создания сетевых соединений. Он удобен для отладки, анализа сети и многих других целей. Так, `netcat` может напрямую обращаться к любой службе TCP или UPD, например SSH-серверу (если он запущен) на локальном TCP-порте 22:

```
→ netcat localhost 22
SSH-2.0-OpenSSH_8.9p1 Ubuntu-3ubuntu0.1
^C
```

Эта функция помогает определить, работает определенная служба или нет. Команда также работает с именами серверов, перечисленными в файле `/etc/services`. Например, вы можете подключиться к веб-службе Google (порт 80):

```
→ netcat www.google.com http
abc <Enter>                    Напечатайте что-нибудь и нажмите Enter
HTTP/1.0 400 Bad Request
Content-Type: text/html; charset=UTF-8
Content-Length: 1555
Date: Tue, 18 Jul 2023 03:14:36 GMT
⋮
```

Олдскулы Linux для подключения к TCP-портам используют `telnet`, но инструмент `netcat` более гибок. Например, вы можете создать клиент и службу, которые будут взаимодействовать друг с другом. Начните со службы, читающей порт 55555:

```
→ netcat -l 55555
```

В другом окне запустите клиент, который будет взаимодействовать с тем же портом, и введите сообщение:

```
→ netcat localhost 55555
Hello world, how are you? <Enter>
```

Ваше сообщение будет передано службе, которая выведет текст «Hello world, how are you?» и все последующие строки, которые вы вводите. Нажмите ^C в клиенте, чтобы закрыть соединение.

Полезные параметры

`-u` Установка UDP-соединения вместо TCP

`-l` Прослушивание соединений на указанном порте

`-p N` Использование порта *N* в качестве источника

`-w N` Тайм-аут через *N* секунд

`-h` Получение справочной информации

Распространенные операции с электронной почтой

`mutt` Текстовый почтовый клиент

`mail` Аскетичный текстовый почтовый клиент

`mailq` Просмотр очереди исходящей почты в системе

Большинство пользователей получают электронную почту в облаке и читают ее в веб-браузере или графическом почтовом приложении. Мало кто читает почту в командной строке, однако текстовые почтовые программы имеют интересные применения, особенно в сценариях. Я расскажу об одном полнофункциональном почтовом клиенте (`mutt`), работающем в командной строке, а также о нескольких других командах, связанных с почтой. Полагаю, что ваша система уже настроена на прием электронной почты, в противном случае некоторые программы не будут работать.

mutt

stdin **stdout** -file --opt --help --version

```
mutt [параметр(ы)]
```

`mutt` — это текстовый почтовый менеджер, запускаемый в оболочке. Вы можете использовать его как локально (в окне терминала), так и удаленно через SSH-соединение. Чтобы вызвать менеджер, выполните команду

```
→ mutt
```

На главном экране `mutt` выводятся сообщения, находящиеся в вашем почтовом ящике, по одному в каждой строке. Попробуйте нажать следующие клавиши.

Клавиша	Значение
↑	Переход к предыдущему сообщению
↓	Переход к следующему сообщению
Page Up	Прокрутка вверх на одну страницу сообщений
Page Down	Прокрутка вниз на одну страницу сообщений
Home	Переход к первому сообщению
End	Переход к последнему сообщению
m	Создание нового сообщения (с вызовом текстового редактора). После редактирования сообщения и закрытия редактора введите y, чтобы отправить сообщение, или q, чтобы сохранить его как черновик
r	Ответ на текущее сообщение. По аналогии с `m`
f	Пересылка текущего сообщения. По аналогии с `m`
i	Просмотр содержимого почтового ящика
C	Копирование текущего сообщения в другой почтовый ящик
d	Удаление текущего сообщения

После завершения редактирования сообщения попробуйте выполнить следующие действия.

Клавиша	Значение
a	Добавление вложения к сообщению
c	Выбор адресатов копии
b	Выбор адресатов скрытой копии
e	Редактирование сообщения
r	Редактирование адреса ответа
s	Редактирование темы
y	Отправка сообщения
C	Копирование сообщения в файл
q	Сохранение сообщения как черновика без отправки

Следующие команды доступны всегда.

Клавиша	Значение
?	Просмотр списка всех команд (нажмите клавишу Пробел, чтобы прокрутить список вниз, q — чтобы выйти)
^G	Отмена выполняемой команды
q	Выход из программы

mail

stdin **stdout** -file --opt --help --version

```
mail [параметр(ы)] получатель
```

Команда `mail` — это простой почтовый клиент. Он лучше всего подходит для отправки быстрых сообщений из командной строки или в сценариях. Например, отправьте сообщение:

```
→ mail smith@example.com
Subject: my subject
I'm typing a message.
To end it, I type a period by itself on a line.
.
Cc: jones@example.com
→
```

Направьте вывод любой команды в `mail`, чтобы отправить его в электронном сообщении. Эта функция особенно полезна в сценариях оболочки:

```
→ echo "Wake up!" | mail -s "Alert" smith@example.com
```

Чтобы отправить содержимое текстового файла по почте, вам нужно передать его на `mail`. Способ не сработает с двоичными файлами, например изображениями, — сначала их нужно преобразовать во вложения.

```
→ mail -s "my subject" smith@example.com < file.txt
```

Полезные параметры

`-s` *тема*	Указание *темы* исходящего сообщения
`-c` *адрес*	Отправка копий на *адреса* из списка, разделенного запятыми
`-b` *адрес*	Отправка скрытых копий на *адреса* из списка, разделенного запятыми
`-v`	Режим подробной информации — вывод сообщений о доставке почты

mailq

stdin **stdout** -file --opt --help --version

```
mailq
```

Команда `mailq` выводит список исходящих сообщений электронной почты, ожидающих доставки (если таковые имеются):

```
→ mailq
... Size-- ----Arrival Time--  -Sender/Recipient---
      333 Tue Jan 10 21:17:14 smith@example.com
                              jones@elsewhere.org
```

Доставка почты часто происходит очень быстро, так что `mailq` даже не имеет вывода. Отправленные сообщения записываются в файл журнала, например `/var/log/mail.log`. В разных дистрибутивах журналы могут называться по-разному. Посмотрим на несколько последних строк с помощью команды `tail`:

```
→ tail /var/log/mail.log
```

За пределами почтовых программ

Электронная почта в Linux функционирует прозрачнее, чем в других платформах, которые просто отображают почтовый ящик, получают и принимают сообщения. Возможность вывода списка исходящих сообщений с помощью `mailq` — только один из примеров. Вот другие возможности, которые побудят вас к исследованиям.

- Можно обрабатывать почтовые ящики с помощью любых инструментов командной строки, например `grep`, так как почтовые файлы — это обычный текст.
- Можно получать сообщения с удаленного почтового сервера вручную из командной строки с помощью команды `fetchmail`. Она позволяет обращаться к серверам IMAP и POP и загружать почту в пакетном режиме (см. `man fetchmail`).
- Можно запустить почтовый сервер, например `postfix`, для комплексной доставки почты (см. раздел «Серверы электронной почты» далее).
- Можно тонко управлять доставкой локальной почты с помощью команды `procmail`, фильтрующей поступающие сообщения электронной почты через указанную программу (см. `man procmail`).
- Можно фильтровать спам с помощью набора программ SpamAssassin. Запускайте программу только для входящей электронной почты или для сервера с большим количеством пользователей.

Электронная почта не ограничивается возможностями вашей почтовой программы. Исследуйте и экспериментируйте!

Серверы электронной почты

Настройка почтового сервера — это сложная работа, которую невозможно освоить, прочитав пару страниц. Я расскажу вам о нескольких общих операциях. Полагаю, что сервер Postfix уже запущен на вашем локальном хосте. Если у вас все еще нет почтового сервера, можете установить

Nullmailer — простой сервер для передачи почты с сервера на сервер.

Postfix — полнофункциональный почтовый сервер

Postfix — это мощный и популярный почтовый сервер, управляемый с помощью команды `postfix`. Важные файлы конфигурации для сервера находятся в каталоге `/etc/postfix`.

`main.cf`
: Содержит переменные, отвечающие за настройки Postfix, такие как имя и домен вашего сервера, расположение важных файлов, ограничения размера почтовых ящиков и входящей почты и многое другое. Изменив этот файл, выполните команду `sudo postfix reload`, чтобы изменения вступили в силу.

`master.cf`
: Определяет, как Postfix запускает различные службы. Обычно пользователи не редактируют этот файл, но если вы сделали это, то выполните команду `sudo postfix reload`, чтобы изменения вступили в силу.

`postfix-files`
: Определяет корректные права доступа для всех файлов Postfix. Эти права очень важны для обеспечения безопасности электронной почты. Обычно пользователи не редактируют этот файл. Чтобы восстановить установку Postfix с этими правами доступа, выполните команду `sudo postfix set-permissions`.

`sasl_passwd`
: Информация об аутентификации для подключения к удаленному SMTP-провайдеру. После изменения этого файла выполните команду `sudo postmap /etc/postfix/sasl_passwd`, чтобы изменения вступили в силу. Формат файла следующий:

  `[smtp-сервер]:`*`порт пользователь:пароль`*

где указываются удаленный *SMTP-сервер*, номер его TCP-*порта* (необязательно) и учетные данные для аутентификации — *пользователь* и *пароль*, например:

```
[smtp.example.com]:587 smith:SEEKRIT_PASSWURD
```

ВНИМАНИЕ!

Защитите все файлы в каталоге /etc/postfix. Файлы sasl_passwd и sasl_passwd.db должны быть доступны для чтения только суперпользователями (режим 0600), так как они содержат пароли.

Обычно сервер Postfix запускается автоматически, но вы можете запускать и останавливать его вручную:

```
→ sudo postfix start      Запуск сервера
→ sudo postfix status     Проверка того, запущен ли сервер
→ sudo postfix stop       Отключение сервера
→ sudo postfix abort      Мгновенное отключение сервера
```

Если у вас возникла проблема с доставкой или отправкой почты, находящейся в очереди, выполните следующую команду:

```
→ sudo postfix flush
```

Дополнительную информацию о сервере Postfix можно найти на сайте postfix.org (https://oreil.ly/NzFRZ).

Nullmailer — простой почтовый сервер

Postfix — сложная система. В Linux есть более простое решение — вы можете перенаправлять почту с локального узла на другой почтовый сервер. (Вам понадобится учетная запись на удаленном почтовом сервере.) Это так называемая настройка почтового сервера только для пересылки писем (Relay-сервер).

Предположим, у вас есть учетная запись `smith` на хосте `example.com` и вы хотите настроить его на отправку электронной почты. У вас есть также учетная запись `sandy` на удаленном сервере `mail.example.com`, на котором уже запущен сервер Postfix для доставки почты. С помощью

менеджера пакетов установите Nullmailer на локальный хост (можете взять его с сайта https://oreil.ly/82aMq). Затем от имени суперпользователя создайте три локальных файла в каталоге `/etc/nullmailer`.

`adminaddr`
: Хранит адрес электронной почты для получения административных писем, например уведомлений Nullmailer.

`defaultdomain`
: Содержит домен, который необходимо установить для всех исходящих сообщений электронной почты.

`remotes`
: Содержит информацию для входа на удаленный почтовый сервер, включая пароль в виде обычного текста, так что проверьте, чтобы этот файл был доступен для чтения только суперпользователем:

```
→ chmod 0600 /etc/nullmailer/remotes
```

Вот так будут выглядеть эти три файла при типичной установке:

```
→ cd /etc/nullmailer
→ cat adminaddr
smith@example.com
→ cat defaultdomain
example.com
→ cat remotes
cat: remotes: Permission denied
→ sudo cat remotes                    Содержимое должно быть в одной строке
mail.example.com smtp --port=587 --tls --user=sandy
  --pass=...
```

В файле `remotes` должны быть указаны значения, характерные для вашего удаленного почтового сервера. Запустите команду `man nullmailer-send`, чтобы узнать их. Когда все три файла будут созданы, включите и запустите службу Nullmailer:

```
→ sudo systemctl enable nullmailer
→ sudo systemctl start nullmailer
```

Отправьте текстовое сообщение:

```
→ echo hi | mail funkydance@another.example.com
```

Проверьте, что сообщение передано на сервер `mail.example.com`:

```
→ tail /var/log/mail          Или другой файл журнала электронной почты
...nullmailer-send: Starting delivery:
                    host: mail.example.com
...nullmailer-send: From: <smith@example.com> to:
     <funkydance@another.example.com>
...nullmailer-send: Delivery complete
```

Просмотр веб-страниц

`lynx`	Текстовый веб-браузер
`curl`	Доступ к онлайн-контенту из командной строки
`wget`	Загрузка веб-страниц и файлов

Помимо привычных веб-браузеров типа Chrome и Firefox, Linux предлагает несколько способов исследования Всемирной паутины с помощью командной строки.

lynx

stdin stdout -file **--opt --help --version**

```
lynx [параметр(ы)] [URL-адрес]
```

Lynx — это облегченный текстовый браузер. Он не показывает изображения, не воспроизводит аудио и видео. Все действия выполняются с помощью клавиатуры, а не мыши. Но Lynx невероятно полезен, когда нужно быстро просмотреть страницу при медленном соединении или загрузить HTML-файл с сайта. Таким образом можно проверять подозрительные URL-адреса, так как Lynx не запускает JavaScript и не принимает cookie, не спросив пользователя:

```
→ lynx https://danieljbarrett.com
```

Многие страницы будут выглядеть очень странно, особенно если на них опубликованы таблицы или изображения, но вы все равно сможете работать с сайтом.

Клавиша	Значение
?	Открытие справки
k	Вывод сочетаний клавиш с описаниями
^G	Отмена выполняемой команды
q	Завершение работы Lynx
↓	Переход к следующей ссылке или полю формы
↑	Переход к предыдущей ссылке или полю формы
Enter	Переход по текущей ссылке или полю формы
→	Переход на следующую страницу
←	Переход на предыдущую страницу
g	Переход по URL-адресу (будет предложено ввести адрес)
p	Сохранение, печать или отправка по почте текущей страницы
Пробел	Прокрутка страницы вниз
b	Прокрутка страницы вверх
^A	Возврат к началу страницы
^E	Возврат к концу страницы
m	Возврат на домашнюю страницу
/	Поиск текста на странице
a	Создание закладки на текущую страницу
v	Просмотр списка закладок
r	Удаление закладки
=	Отображение свойств текущей страницы и ссылки
\	Просмотр исходного HTML-кода (нажмите еще раз, чтобы вернуться к обычному просмотру)

В Lynx используется более 100 параметров, так что рекомендую прочитать документацию. Другие текстовые браузеры — это w3m, links и elinks.

Полезные параметры

`-dump` Передача визуализированной страницы на стандартный вывод (сравните с параметром `-source`)

`-source` Передача исходного HTML-кода на стандартный вывод (сравните с командой `wget`)

`-useragent=`*`имя`*	Если сайт блокирует `lynx`, можно указать *имя* пользовательского агента для имитации другого браузера, например `-useragent=mozilla`
`-emacskeys`	Использование в Lynx сочетаний клавиш, как в emacs
`-vikeys`	Использование в Lynx сочетаний клавиш, как в Vim
`-homepage=`*`URL-адрес`*	Установка *URL-адреса* домашней страницы
`-color`	Включение режима цветного текста
`-nocolor`	Отключение режима цветного текста

curl

stdin stdout -file --opt --help --version

```
curl [параметр(ы) | URL-адрес]
```

Команда `curl` открывает доступ к онлайн-контенту. Она может загружать веб-страницы, проверять веб-службы и выполнять любые операции с URL-адресами. По умолчанию `curl` передает результат на стандартный вывод, но вы можете перенаправлять его по своему усмотрению или с помощью параметра `-o`:

```
→ curl https://www.yahoo.com > mypage.html      Перенаправление
→ curl -o mypage.html https://www.yahoo.com     Параметр -o
→ curl https://www.yahoo.com | wc -l            Конвейер
```

Команда `curl` подходит для быстрого тестирования REST API. В «Википедии» выполните поиск по слову *Linux* — получите ответ в формате JSON:

```
→ curl "https://en.wikipedia.org/w/rest.php/v1/search/ \
  page?q=Linux"
{"pages":[{"id":6097297,"key":"Linux","title":"Linux" ...
```

У команды десятки параметров и имеется поддержка множества протоколов помимо HTTP, например IMAP, FTP и SMB. Команда может добавлять веб-заголовки, размещать данные на веб-страницах, аутентифицировать пользователей по паролю, работать с SSL-сертификатами, имитировать cookie-файлы и многое другое (см. документацию).

Полезные параметры

`-o` *файл*	Запись вывода в указанный *файл*
`-A` *имя*	Указание *имени* пользовательского агента для имитации другого браузера. Попробуйте ввести `-A mozilla`
`-v`	Подробный режим — вывод подробной информации для диагностики
`-s`	Краткий режим — вывод для диагностики отсутствует

wget

stdin stdout -file --opt --help --version

`wget [`*`параметр(ы)`*`] `*`URL-адрес`*

Команда `wget` загружает данные с *`URL-адреса`* в файл. Она отлично подходит для извлечения отдельных веб-страниц, загрузки файлов и дублирования целых иерархий веб-сайтов. Например, давайте соберем данные с домашней страницы Yahoo!:

```
→ wget https://www.yahoo.com
2023-10-22 13:56:53 (3.45 MB/s) - 'index.html' saved
```

Данные сохранятся в файл `index.html`, находящийся в текущем каталоге. Команда `wget` может возобновить загрузку, прерванную, например, из-за сбоя в сети: просто запустите `wget -c` с тем же URL-адресом, и загрузка начнется с того места, на котором остановилась.

Пожалуй, самая полезная функция `wget` — это возможность загружать файлы без браузера:

```
→ wget https://linuxpocketguide.com/sample.pdf
```

Это удобно во время работы с большими файлами, например видеороликами и ISO-образами. Вы можете писать сценарий оболочки для загрузки наборов файлов (если знаете имена файлов). Далее показан пример сценария, загружающего три видеофайла в формате MP4 с именами `1.mp4`, `2.mp4` и `3.mp4` из корня веб-сайта:

```
→ for i in 1 2 3
do
  wget https://example.com/$i.mp4
done
```

Команда `wget` поддерживает свыше 70 параметров, так что я рассмотрю только самые важные из них.

Полезные параметры

`-U` *имя*	Если сайт блокирует `wget`, можно указать *имя* пользовательского агента для имитации другого браузера. Например, попробуйте `-U mozilla`
`-O` *файл*	Запись всех HTML-данных в указанный *файл*
`-i` *файл*	Чтение URL-адресов из указанного *файла* и поочередное извлечение данных
`-c`	Продолжение прерванной загрузки. Например, если команда `wget` загрузила 100 Кбайт из 150 Кбайт, то с командой `wget -c` будут загружены только оставшиеся 50 Кбайт. Однако используйте `-c` только в том случае, если удаленный файл не изменился с момента первой загрузки, иначе программа `wget` не сможет загрузить его полностью
`-t` *N*	Выполнение *N* попыток операции. Если указано значение 0, выполняется бесконечное количество попыток
`--progress=dot`	Вывод точек вместо полос для отображения процесса загрузки
`--spider`	Без загрузки, только проверка существования удаленных страниц
`-nd`	Загрузка всех файлов в текущий каталог вместо дублирования иерархии удаленных каталогов
`-r`	Рекурсивное извлечение всего дерева страниц
`-l` *N*	Извлечение файлов с глубиной вложенности не более чем на *N* уровней (по умолчанию 5)
`-k`	Изменение URL-адресов в содержимом извлеченных файлов так, чтобы файлы можно было просматривать локально
`-p`	Загрузка всех необходимых файлов для полного отображения страницы, например таблиц стилей и изображений
`-L`	Отслеживание относительных, а не абсолютных ссылок (в пределах страницы)
`-A` *шаблон*	Белый список файлов — загрузка только тех, имена которых *соответствуют* указанному *шаблону* с использованием стандартного шаблона оболочки
`-R` *шаблон*	Черный список файлов — загрузка только тех, имена которых *не соответствуют* указанному *шаблону*
`-I` *шаблон*	Белый список каталогов — загрузка файлов только из каталогов, *соответствующих* указанному *шаблону*
`-X` *шаблон*	Черный список каталогов — загрузка файлов только из каталогов, *не соответствующих* указанному *шаблону*

ГЛАВА 6

Решение обыденных задач

Вывод на экран

`echo`	Передача простого текста на стандартный вывод
`printf`	Передача отформатированного текста на стандартный вывод
`yes`	Передача повторяющегося текста на стандартный вывод
`seq`	Передача последовательности чисел на стандартный вывод
`clear`	Очистка экрана или окна

В Linux есть несколько команд для печати сообщений на стандартный вывод. Каждая из них имеет свои особенности и предназначение. Эти команды помогают при изучении Linux, отладке, написании сценариев оболочки (см. «Программирование с помощью сценариев командной оболочки» далее в данной главе) или просто при общении с самим собой.

echo

stdin **stdout** -file --opt --help --version

`echo [`*`параметр(ы)`*`] `*`строка`*

Команда `echo` просто выводит переданный аргумент:

```
→ echo We are having fun
We are having fun
```

Иногда возникает путаница, так как есть несколько разных команд `echo` с немного различающимся поведением. Существует `/bin/echo`, но оболочки Linux обычно заменяют ее встроенной командой `echo`. Чтобы узнать, какой

вариант команды вы используете, выполните следующую команду:

```
→ type echo
echo is a shell builtin
```

Полезные параметры

`-a` Без вывода конечного символа перевода строки

`-e` Обработка и интерпретация управляющих символов. Например, попробуйте выполнить команды `'hello\a'` и `echo -e 'hello\a'`. Первая выводит текст дословно, а вторая издает звуковой сигнал

`-E` Без интерпретации управляющих символов. Противоположность параметру `-e`

Допустимы следующие управляющие символы.

`\a` Внимание! (воспроизведение звукового сигнала)

`\b` Возврат на шаг

`\c` Без вывода конечного символа перевода строки (то же самое, что и `-n`)

`\f` Формат вывода

`\n` Перевод строки

`\r` Возврат каретки

`\t` Горизонтальный отступ

`\v` Вертикальный отступ

`\\` Обратный слеш

`\'` Одиночная кавычка

`\"` Двойная кавычка

`\nnn` Символ, значение ASCII которого равно *nnn* в восьмеричной системе (основание 8)

printf

stdin **stdout** -file **--opt** **--help** --version

```
printf строка [аргумент(ы)]
```

Команда `printf` — это продвинутая команда `echo`: она печатает отформатированные *строки* в стандартном выводе. Работает как функция языка программирования C `printf()`, которая применяет f-строку к последовательности аргументов для формирования вывода, например:

```
→ printf "User %s is %d years old.\n" sandy 29
User sandy is 29 years old.
```

Первый аргумент — это f-строка с двумя спецификациями формата, `%s` и `%d`. Следующие аргументы, `sandy` и `29`, подставляются `printf` в f-строку, а затем выводятся. Если вы работаете с числами с плавающей точкой, то спецификации формата могут выглядеть странно:

```
→ printf "That'll be $%0.2f, my friend.\n" 3
That'll be $3.00, my friend.
```

В Linux есть две команды `printf`: одна встроена в bash, а другая находится в `/usr/bin/printf`. Они работают практически одинаково. Различия невелики — например, только встроенная команда `printf` может сохранять вывод в переменной оболочки (с параметром `-v`).

Убедитесь, что количество спецификаций формата (`%`) равно количеству аргументов `printf`. Если аргументов слишком мало, то `printf` выводит значения по умолчанию (`0` для числовых форматов, пустую строку для строковых форматов). Если аргументов слишком много, то `printf` перебирает дополнительные аргументы, пока они не закончатся. В принципе, это ошибки, хотя `printf` и не предупреждает о них:

```
→ printf "%d %d\n" 10                    Слишком мало аргументов
10 0
→ printf "%d %d\n" 10 20 30 40           Слишком много аргументов
10 20
30 40
```

Подробно о спецификации формата можно прочитать в документации к функции `printf` языка C (см. `man 3 printf`). В следующей таблице я привел самые полезные из них.

`%d`	Целое десятичное число
`%ld`	Длинное целое десятичное число
`%o`	Восьмеричное целое число
`%x`	Шестнадцатеричное целое число
`%f`	Плавающая запятая
`%lf`	Двойной точности с плавающей точкой

`%c`	Одиночный символ
`%s`	Строка
`%q`	Строка с экранированными специальными символами оболочки
`%%`	Символ процента

Вы можете ограничить минимальную ширину вывода, добавив числовое выражение после ведущего знака процента. Например, `%5d` означает вывод десятичного числа в поле шириной пять символов, а `%6.2f` — вывод числа с плавающей точкой в поле шириной шесть символов и двумя числами после запятой. К числу полезных числовых выражений можно отнести следующие:

n	Минимальная ширина *n*
0n	Минимальная ширина *n*, заполненная ведущими нулями
n.m	Минимальная ширина *n* с *m* цифрами после запятой

Команда `printf` также умеет интерпретировать такие символы, как `\n` (перевод строки) и `\a` (звуковой сигнал). Полный список символов приводится в документации к команде `echo`.

---

yes

stdin **stdout** -file --opt **--help** **--version**

```
yes [строка]
```

Команда `yes` постоянно выводит символ `y` или указанную *строку*, по одной букве в строке:

```
→ yes
y
y
y
⋮
```

На первый взгляд эта команда бесполезна, но она может подойти для команд, которые предлагают пользователю продолжить действие. Подключите команду `yes` к выводу команды, чтобы утвердительно отвечать на каждый запрос:

```
→ yes | интерактивная_команда
```

Когда *интерактивная_команда* завершается, заканчивается и работа команды `yes`.

---

seq

stdin **stdout** -file --opt **--help** **--version**

```
seq [параметр(ы)] число
```

Команда `seq` выводит последовательность целых или вещественных *чисел*, которую можно передать в другие программы. Она поддерживает три вида аргументов.

Одно число — верхний предел

`seq` начинает счет с 1 и продолжает его до указанного числа:

```
→ seq 3
1
2
3
```

Два числа — верхний и нижний пределы

`seq` начинает счет с первого числа и перечисляет все числа до второго:

```
→ seq 2 5
2
3
4
5
```

Три числа — нижний предел, приращение и верхний предел

`seq` начинает счет с первого числа и увеличивается на второе число столько раз, чтобы дойти до третьего числа:

```
→ seq 1 .3 2
1
1.3
1.6
1.9
```

Умеет она и отсчитывать в обратном порядке с отрицательным приращением:

```
→ seq 5 -1 2
5
4
3
2
```

Полезные параметры

`-w` Вывод ведущих нулей, чтобы сделать все строки одинаковой ширины:

```
→ seq -w 8 10
08
09
10
```

`-f` *формат* Форматирование результирующих строк согласно *формату* по аналогии с `printf`, которая должна включать либо `%g` (по умолчанию), либо `%e`, либо `%f`:

```
→ seq -f '**%g**' 3
**1**
**2**
**3**
```

`-s` *строка* Применение указанной *строки* в качестве разделителя между числами. По умолчанию передается символ перевода строки (то есть выводится одно число в строке):

```
→ seq -s ':' 10
1:2:3:4:5:6:7:8:9:10
```

Bash и другие оболочки имеют аналогичную функцию для создания последовательности чисел (см. раздел «Использование скобок» главы 1):

```
→ echo {1..10}
1 2 3 4 5 6 7 8 9 10
```

clear

stdin **stdout** -file --opt --help --version

```
clear
```

Команда `clear` удаляет информацию с экрана или из окна оболочки. В качестве альтернативы нажмите клавишу ^L.

Копирование и вставка

`xclip`	Копирование/вставка данных между оболочкой и буферами
`xsel`	Управление буфером из оболочки

В Linux реализован буфер для копирования/вставки данных между графическими приложениями. Всего в Linux *три* буфера — первичный, вторичный и буфер обмена. Вы можете получить доступ к ним из командной строки, скопировать в буфер вывод любой команды оболочки или извлечь данные из буфера на стандартный ввод.

Эти команды работают только в том случае, если ваша оболочка запущена в среде X11, например GNOME или KDE. В Wayland есть другие механизмы буферов для обмена данными, например `wl-copy` и `wl-paste`. В неграфических средах команды `tmux` и `screen` обеспечивают работу с буферами. Если вы не используете ни одну из этих сред, то буферы работать не будут.

xclip

stdin stdout -file --opt **--help --version**

```
xclip [параметр(ы)]
```

`xclip` считывает и записывает три буфера Linux для копирования/вставки текста между оболочкой и графическими приложениями. Чтобы запустить команду, скопируйте с помощью мыши текст в буфер (дважды щелкните мышью на слове в окне терминала) и запустите команду

```
→ xclip -o
```

Скопированный текст будет передан на стандартный вывод. В качестве другого примера скопируйте содержимое файла в буфер, а затем выведите скопированные данные:

```
→ cat poem                                    Просмотр файла
Once upon a time, there was
a little operating system named
Linux, which everybody loved.
→ cat poem | xclip -i                         Копирование данных в буфер
```

```
→ xclip -o                                    Вывод данных из буфера
Once upon a time, there was
a little operating system named
Linux, which everybody loved.
```

Все параметры командной строки для `xclip` сопровождаются одинарным дефисом, даже `-help` и `-version`.

Полезные параметры

`-selection` *буфер*	Выбор *буфера*: `primary` (первичный, по умолчанию), `secondary` (вторичный) или `clipboard` (обмена). В моем случае в терминале средняя кнопка мыши вставляет данные из первичного буфера, а команда Paste (Вставить) из контекстного меню (меню правой кнопки мыши) вставляет данные из буфера обмена
`-i`	Считывание содержимого буфера со стандартного ввода (по умолчанию)
`-o`	Запись содержимого буфера в стандартный вывод

---

xsel

stdin **stdout** -file --opt **--help** **--version**

```
xsel [параметр(ы)]
```

`xsel` — это наиболее мощная версия `xclip`. Кроме чтения данных из трех буферов и записи в них, команда может добавлять данные, менять их местами или удалять:

```
→ echo Hello | xsel -i
→ xsel -o
Hello
→ echo World | xsel -a                        Добавление
→ xsel -o
Hello
World
```

Полезные параметры

`-p`	Использование основного буфера (по умолчанию)
`-s`	Использование вторичного буфера
`-b`	Использование буфера обмена
`-i`	Чтение содержимого буфера со стандартного ввода (по умолчанию)

`-a`	Добавление в буфер
`-o`	Запись содержимого буфера в стандартный вывод
`-c`	Очистка содержимого буфера
`-x`	Обмен содержимого первичного и вторичного буферов

Математические операции и вычисления

`expr`	Выполнение простых математических вычислений в командной строке
`bc`	Текстовый калькулятор
`dc`	Текстовый постфиксный калькулятор

Нужен калькулятор? В Linux есть несколько команд для вычисления математических уравнений.

---

expr

stdin **stdout** -file --opt **--help** **--version**

`expr` *выражение*

Команда `expr` выполняет простые математические вычисления в командной строке:

```
→ expr 7 + 3
10
→ expr '(' 7 + 3 ')' '*' 14  Спецсимволы следует указывать в кавычках
140
→ expr length ABCDEFG
7
→ expr 15 '>' 16
0                                  Ноль = ложь
```

В bash используется сокращение для `expr` — знак доллара и двойные круглые скобки — `$((…))`. Это очень полезный прием для вычислений в командной строке:

```
→ echo The answer is: $((7 + 3))
The answer is: 10
```

Аргументы должны быть разделены пробелами. Обратите внимание на то, что все символы, имеющие специальное значение для оболочки, должны быть заключены в кавычки или экранированы. Для группировки можно использовать

круглые скобки (экранированные). Операторы для `expr` перечислены в таблице далее.

Оператор	Числовые операции	Строковая операция
`+`, `-`, `*`, `/`	Сложение, вычитание, умножение и деление соответственно	
`%`	Остаток	
`<`	Меньше	Ранее в словаре
`<=`	Меньше чем или равно	Ранее в словаре или равно
`>`	Больше	Позже в словаре
`>=`	Больше чем или равно	Позже в словаре или равно
`=`	Равно	Равно
`!=`	Не равно	Не равно
`|`	Логическое ИЛИ	Логическое ИЛИ
`&`	Логическое И	Логическое И
`s:`*`шаблон`*		Совпадает ли шаблон со строкой *s*?
`substr` *s p n*		Вывод *n* символов строки *s*, начиная с позиции *p* (первый символ — это `p=1`)
`index` *s строка*		Вывод индекса первой позиции в строке *s*, содержащей символ из *строки*. Если вывод не найден, возвращает `0`

Для логических выражений число 0 или пустая строка считаются ложными, а любое другое значение — истинным. При возврате булевых значений `0` — это ложь, а `1` — истина.

bc

stdin stdout -file **--opt --help --version**

```
bc [параметр(ы)] [файл(ы)]
```

`bc` — это текстовый калькулятор, считывающий по одному арифметическому выражению в строке и выводящий результат. В отличие от большинства других калькуляторов,

bc может работать с числами неограниченного размера и с любым количеством десятичных знаков:

```
→ bc
1+2+3+4+5                          Сложение пяти чисел
15
scale=2                            Ограничение до двух знаков после запятой
(1 + 2 * 3 / 4) - 5
-2.50
2^100                              2 в степени 100
1267650600228229401496703205376
^D                                 Завершение работы
```

Программистам понравится возможность переключения оснований для выполнения вычислений в двоичной, восьмеричной, шестнадцатеричной или даже пользовательской системе:

```
→ bc
obase=2                            Основание 2
999
1111100111
obase=16                           Основание 16
999
3E7
```

Этим возможности bc не ограничиваются. Ведь это программируемый калькулятор, который может определять функции. Так, далее показана функция, принимающая файл quadratic.txt и использующая квадратическую алгебраическую формулу с выводом действительных корней указанного уравнения[1]:

```
→ cat quadratic.txt
scale=2
define quadform ( a, b, c ) {
 root1 = (-b + sqrt(b^2 - 4*a*c)) / (2*a)
 root2 = (-b - sqrt(b^2 - 4*a*c)) / (2*a)
 print root1, "  ", root2, "\n"
}

quadform(1, 7, 12)                 Решение: x² + 7x + 12 = 0
```

[1] Если корни мнимые, то вариант из примера не сработает.

Перенаправьте файл команде bc, чтобы запустить функцию и посмотреть результаты:

```
→ bc < quadratic.txt
    -3.00 -4.00
```

В самом мощном проявлении bc — это язык программирования для арифметических выражений. Вы можете присваивать значения переменным, работать с массивами, выполнять условные выражения и запускать циклы. Можете даже писать сценарии, запрашивающие значения и выполняющие любую последовательность математических операций. Более подробную информацию ищите в документации.

Полезные математические операции

+, -, *, /	Сложение, вычитание, умножение и деление соответственно. Результаты деления обрезаются до определенного количества цифр
%	Остаток
^	Возведение в степень. Например, 10^5 — десять в пятой степени
sqrt(*N*)	Квадратный корень из *N*
ibase=*N*	Обработка всех чисел с основанием *N*
obase=*N*	Вывод всех чисел в основании *N*
scale=*N*	Указание *N* цифр после десятичной запятой
(…)	Скобки для группирования (изменения приоритета вычислений)
переменная=значение	Присвоение *значения переменной*

dc

stdin stdout -file --opt --help --version

dc [*параметр(ы)*] [*файл(ы)*]

Команда dc — это постфиксный (работающий в режиме обратной польской записи, RPN) текстовый калькулятор, считывающий выражения со стандартного ввода и записывающий результаты в стандартный вывод. Если вы умеете пользоваться RPN-калькулятором Hewlett-Packard, то

команда `dc` не вызовет у вас никаких трудностей. Но если вы привыкли к традиционным калькуляторам, то освоить `dc` может оказаться очень нелегко. Я рассматриваю некоторые основные команды.

Для операций со стеком и вычислений:

`q`	Завершение работы `dc`
`f`	Вывод стека
`c`	Очистка стека
`p`	Вывод верхнего значения
`P`	Удаление верхнего значения
n `k`	Определение точности будущих операций в *n* десятичных знаков (по умолчанию `0k`, что означает операции с целыми числами)

Чтобы извлечь два верхних значения из стека, выполните нужную операцию и вставьте результат.

`+, -, *, /`	Сложение, вычитание, умножение и деление соответственно
`%`	Остаток
`^`	Возведение в степень (второе значение — основание, верхнее значение — степень)

Чтобы извлечь верхнее значение из стека, выполните нужную операцию и вставьте результат.

`v`	Квадратный корень

Примеры:

```
→ dc
4 5 + p          Вывод суммы чисел 4 и 5
9
2 3 ^ p          Возведение числа 2 в степень 3 и вывод результата
8
10 * p           Умножение верхнего значения стека на 10 и вывод результата
80
f                Вывод стека
80
9
+p               Сложение двух верхних значений и вывод результата
89
^D               Завершение работы
```

Дата и время

`cal` Вывод календаря

`date` Вывод или установка даты и времени

Нужно назначить свидание? Или запланировали хорошо провести время? Следующие команды помогут вам отобразить и установить дату и время в своей системе.

---

cal

stdin **stdout** -file --opt --help --version

```
cal [параметр(ы)] [месяц [год]]
```

Команда `cal` выводит календарь (по умолчанию на текущий месяц):

```
→ cal
    January 2024
Su Mo Tu We Th Fr Sa
    1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31
```

Чтобы вывести календарь на другой период, укажите *месяц* и четырехзначный *год*: `cal 8 2024` или `cal aug 2024`. Если вы опустите месяц (`cal 2024`), то будет выведен календарь на весь год. Обратите внимание на то, что существует несколько разных программ `cal`, которые могут работать по-разному.

Полезные параметры

`-y` Вывод календаря на текущий год

`-3` Квартальный отчет — вывод предыдущего и следующего месяцев

`-j` Вывод дат в соответствии с их позициями в году. 1 февраля будет отображено как 32, 2 февраля — как 33 и т. д.

date

stdin **stdout** -file --opt **--help** **--version**

```
date [параметр(ы)] [формат]
```

Команда `date` выводит дату и время. Результаты зависят от настроек локализации вашей системы (от ваших страны и языка). В этом разделе я использую американскую английскую локализацию.

По умолчанию `date` выводит системную дату и время, соответствующее местному часовому поясу:

```
→ date
Sun Jun  4 02:09:05 PM EDT 2023
```

Вы можете отформатировать вывод, указав f-строку, начинающуюся со знака «плюс»:

```
→ date +%x
06/04/2023
→ date '+The time is %l:%M %p on a lovely %A in %B'
The time is  2:09 PM on a lovely Sunday in June
```

Перечислю некоторые из многочисленных форматов команды `date`.

Формат	Значение	Пример (американский английский)
Дата и время		
%c	Полная дата и время, 12-часовой формат	Sun 04 Jun 2023 02:09:05 PM EDT
%D	Числовая дата, двузначный год	06/04/23
%x	Числовая дата, четырехзначный год	06/04/2023
%T	Время, 24-часовой формат	14:09:05
%X	Время, 12-часовой формат	02:09:05 PM
Слова		
%a	День недели (сокращенно)	Sun
%A	День недели (полностью)	Sunday
%b	Название месяца (сокращенно)	Jun

Формат	Значение	Пример (американский английский)
%B	Название месяца (полностью)	June
%z	Часовой пояс	EDT
%p	Половина дня (AM и PM до 12-часового формата)	PM
Числа		
%w	День недели (0–6, 0 — воскресенье)	0
%u	День недели (1–7, 1 — понедельник)	7
%d	День месяца, ведущий ноль	04
%e	День месяца, ведущий пробел	4
%j	День года, ведущий ноль	144
%m	Номер месяца, ведущий ноль	06
%y	Год, 2 цифры	23
%Y	Год, 4 цифры	2023
%M	Минуты, ведущий пробел	09
%S	Секунды, ведущий ноль	05
%l	Час, 12-часовой формат, ведущий пробел	2
%I	Час, 12-часовой формат, ведущий ноль	02
%k	Час, 24-часовой формат, ведущий пробел	14
%H	Час, 24-часовой формат, ведущий ноль	14
%N	Наносекунды	384789400
%s	Секунд с начала эпохи Linux — полночи 1 января 1970 года	1685902145
Другое		
%n	Символ перевода строки	
%t	Символ табуляции	
%%	Символ процента	%

`date` может отображать другие даты и время с помощью параметров.

Полезные параметры

`-d` *строка* — Вывод указанной *строки* даты или времени, отформатированной по вашему желанию

`-r` *файл* — Отображение последней именной метки времени указанного *файла* в нужном формате

`-s` *строка* — Установка системной даты и/или времени согласно *строке*. Это может сделать только суперпользователь

Управление версиями

`git` — Управление версиями с помощью Git

`svn` — Управление версиями с помощью Subversion

Системы управления версиями, например Git и Subversion, позволяют отслеживать изменения в файлах, безопасно отменять рискованные правки и совместно с коллегами редактировать одни и те же файлы. Я предполагаю, что вы уже знакомы с такими понятиями управления версиями, как репозитории, ветви и коммиты.

---

git

stdin **stdout** -file **--opt** **--help** **--version**

`git [`*параметр(ы)*`] `*команда*` [`*аргумент(ы)*`]`

Git — это популярная мощная система управления версиями. Она предоставляет каждому участнику собственный полнофункциональный репозиторий и позволяет делиться изменениями с другими репозиториями. Команда `git` обладает богатым набором функций, который слишком велик, чтобы рассказать о нем здесь, но давайте рассмотрим некоторые общие операции.

Чтобы начать работу, создайте локальный репозиторий:

```
→ mkdir project
→ cd project
→ git init .
```

Чтобы загрузить копию удаленного хранилища по его URL-адресу, выполните команду

```
→ git clone URL-адрес
```

Когда вы отредактируете некоторые файлы, обычная последовательность команд `git` будет выглядеть следующим образом:

```
→ git status                    Вывод списка измененных файлов
→ git diff                      Построчная проверка изменений
→ git add -A                    Подготовка измененных файлов
→ git diff --staged             Проверка изменений
→ git commit -m"комментарий"    Фиксация (коммит) изменений локально
→ git show                      Просмотр зафиксированных изменений
```

Подготовка и отмена изменений

Подготовка означает копирование файлов в скрытую область подготовленных файлов для последующего коммита в `git`. Противоположным действием служит команда `git reset`, удаляющая файлы из этой области:

```
→ git reset файл(ы)    Удаление указанных файлов
→ git reset            Удаление ВСЕХ файлов
```

Вы также можете отменить изменения, которые еще не были помечены как подготовленные, откатив некоторые или все файлы к их последнему коммиту:

```
→ git restore файл(ы)    Откат изменений без пометки о подготовке
→ git reset --hard       Откат ВСЕХ изменений
```

В старых версиях Git вместо команды `git restore` используется `git checkout`.

ВНИМАНИЕ!

Команда git reset --hard удаляет все незафиксированные изменения независимо от того, прошли они подготовку или нет. Будьте осторожны при ее использовании!

Если вы работаете с удаленным репозиторием, то можете отправлять туда сделанные вами изменения в удобное время. Показанная далее команда `git` переносит эти изменения в удаленную ветвь, совпадающую с вашей локальной:

```
→ git push origin HEAD
```

Добавьте изменения, внесенные коллегами по группе, в свой локальный репозиторий. Так можно убедиться, что ваши изменения совпадают с изменениями ваших коллег, прежде чем выкладывать их:

```
→ git pull
```

Другой распространенный рабочий процесс — создание локальной ветви, фиксация в ней различных изменений, их слияние с основной ветвью и удаление локальной ветви по окончании работы[1]:

```
→ git switch -c ветвь          Создание локальной ветви
→ ...                          Изменение файлов и фиксация изменений
→ git switch main              Возвращение в основную ветвь
→ git merge ветвь              Объединение изменений
→ git branch -d ветвь          Удаление локальной ветви
```

Вы также можете перенести изменения в удаленную ветвь, отличную от основной:

```
→ git push origin ветвь        Отправка изменений
```

Когда коллеги закончат работу с удаленной ветвью, вам нужно убрать ее:

```
→ git push origin --delete ветвь    Уничтожение удаленной ветви
```

Можете посмотреть историю изменений в журнале:

```
→ git log
```

У каждого коммита есть идентификатор ревизии — строка из 40 шестнадцатеричных цифр, иногда сокращаемая до первых семи цифр. Многие `git`-команды принимают

[1] В старых версиях Git вместо git switch [-c] используется команда git checkout [-b].

идентификаторы ревизии в качестве аргументов, например `git log b77eb6b`.

Приведу другие распространенные операции:

```
→ git branch                          Перечисление всех ветвей
→ git mv источник назначение          Перемещение файла/каталога
→ git cp источник назначение          Копирование файла/каталога
→ git rm файл                         Удаление файла
→ git rm -r каталог                   Удаление каталога
```

---

svn

stdin **stdout** -file **--opt** **--help** **--version**

```
svn команда [параметр(ы)] [аргумент(ы)]
```

Subversion — это система управления версиями для дерева файлов. В отличие от Git, Subversion требует наличия репозитория, прежде чем вы сможете работать с файлами. Я предполагаю, что у вас уже есть доступ к репозиторию. Команда `svn` имеет богатый набор функций, так что я не буду перечислять их все, а представлю некоторые популярные операции.

Чтобы начать работу, найдите URL-адрес существующего репозитория и проверьте его, создав локальное *рабочее* пространство:

```
→ svn checkout URL-адрес
```

После редактирования файлов в рабочем пространстве выполняется обычная последовательность действий. Команды `svn` выглядят так:

```
→ svn status                          Вывод измененных файлов
→ svn diff                            Просмотр изменений
→ svn commit -m"комментарий"          Фиксация изменений
```

Если вы создаете новые файлы, то перед фиксацией должны сообщить Subversion, что они относятся к вашему рабочему пространству:

```
→ svn add файл                        Добавление в рабочее пространство
→ svn commit -m"комментарий"          Фиксация изменений
```

Добавьте изменения, сделанные коллегами по группе, в свой локальный репозиторий:

```
→ cd корень_вашего_рабочего_пространства
→ svn update
```

Так можно убедиться, что ваши изменения совпадают с изменениями ваших коллег, прежде чем вы их выложите.

Другой распространенный рабочий процесс — создание ветви, фиксация в ней различных изменений, их слияние с основной ветвью и удаление локальной ветви по окончании работы. Синтаксис ветвления в Subversion сложнее, чем в Git. Имя ветви обычно представляет собой URL-адрес в репозитории:

```
https://example.com/путь          Удаленный сервер, SSL
svn://example.com/путь            Удаленный сервер, протокол SVN
svn://localhost/путь              Локальный сервер, протокол SVN
```

Показанные далее команды создают ветвь посредством копирования основной ветви, которую часто называют *стволом*. Затем они фиксируют изменения в новой ветви и возвращают ее обратно в ствол:

```
→ ls
my_project
→ svn cp my_project my_branch              Создание ветви
→ nano my_branch/file.py                   Редактирование файла
→ svn commit -m"edited in a branch"        Фиксирование в ветви
Adding          my_branch
Sending         my_branch/file.py
Transmitting file data .done
Committing transaction...
Committed revision 966.
→ cd my_project
→ svn merge -c966 svn://example.com/my_branch .          Слияние
--- Merging r966 into .:
U    file.py
→ svn commit -m"merging into trunk"          Возвращение в ствол
Sending        file.py
Transmitting file data .done
Committing transaction...
Committed revision 967.
```

Теперь взглянем на историю изменений:

```
→ svn log
```

У каждой фиксации есть идентификатор ревизии — положительное целое число, которое увеличивается на единицу с каждой фиксацией. Идентификаторы отображаются в журнале. Многие команды `svn`, например `svn log -r25`, принимают идентификаторы ревизии в качестве аргументов. Диапазон ревизий ограничивается с помощью двух идентификаторов, разделенных двоеточием, — `svn log -r25:28`.

Другие операции:

```
→ svn mv источник назначение      Перемещение файла/каталога
→ svn rm путь                     Удаление файла/каталога
→ svn cat [-r ревизия] файл       Вывод файла в stdout
→ svn info путь                   Вывод информации о файлах
```

Контейнеры

`docker` Упаковка и запуск приложений в контейнерах

Сталкивались ли вы с ситуацией, когда, устанавливая приложение, вы обнаруживали, что операционная система Linux не настроена для его запуска? Если да, то у меня есть для вас решение. *Контейнеры* — это способ упаковать код приложения и все его зависимости, чтобы оно было готово к запуску на других системах. Каждый запущенный контейнер — это изолированная миниатюрная среда (вроде виртуальной машины), что особенно полезно при масштабировании приложений в облаке.

Docker — это популярная инфраструктура для контейнеров. Я расскажу вам об основных способах ее использования. И конечно, я предполагаю, что у вас уже установлена среда Docker.

docker

stdin **stdout** -file --opt **--help** **--version**

```
docker команда [параметр(ы)] [аргумент(ы)]
```

Команда `docker` создает и запускает контейнеры, а также управляет ими. Давайте начнем с текстового файла `Dockerfile`, который определяет содержимое и поведение запущенного контейнера. Далее показано содержимое простого файла `Dockerfile` для образа, засыпающего на 5 минут, но также способного запускать веб-серверы и выполнять вычисления:

```
# Образ, засыпающий на 5 минут.
# Основан на небольшом образе alpine.
FROM alpine
CMD ["sleep", "300"]
```

Затем скомпилируйте `Dockerfile`, чтобы создать *образ*, готовый к запуску. Наконец, запустите один или несколько экземпляров образа, каждый из которых находится в изолированном контейнере. Типичный рабочий процесс выглядит следующим образом:

```
→ docker build -t example .        Создание образа с именем example
→ docker run example &             Запуск образа в контейнере
→ docker run example &             Запуск образа в другом контейнере
→ docker ps                        Вывод списка запущенных контейнеров
CONTAINER ID   IMAGE     COMMAND       ...
ff6d0f5ad309   example   "sleep 300"   ...
f80519480635   example   "sleep 300"   ...
```

Вы также можете скачать и запустить образ из интернета. Репозиторий Docker Registry — это инфраструктура для обмена образами. Типичный рабочий процесс выглядит следующим образом:

```
→ docker search hello                  Поиск в реестре
NAME          DESCRIPTION
hello-world   Hello World! (an example of minimal...
⋮
→ docker pull hello-world              Загрузка образа
→ docker run hello-world               Запуск образа в контейнере
```

ПРИМЕЧАНИЕ

В зависимости от установки docker вам может потребоваться запускать команды от имени суперпользователя (sudo docker) или присоединиться к группе docker, чтобы делать это от своего имени (sudo usermod -aG docker $USER).

Распространенные команды Docker

Прежде чем выполнять команды, требующие имя или идентификатор контейнера, их нужно найти. Для этого запустите `docker ps`. Большинство подкоманд `docker` принимают параметры. Есть и множество других дополнительных команд, которые я не рассматриваю. На сайте Docker (https://oreil.ly/6eNA6) вы можете найти подробную информацию.

Образы и контейнеры

`docker search` *строка*	Найти в репозитории образ со *строкой* в имени или в описании
`docker pull` *образ*	Загрузка *образа* из репозитория
`docker build` *образ*	Создание *образа* из файла Docker
`docker create` *образ*	Создание контейнера из *образа*, без запуска

Выполнение контейнеров

`docker run` *образ*	Создание контейнера и запуск *образа*
`docker stop` *имя*	Остановка выполнения контейнеров с указанными *именами* или идентификаторами. Контейнеры можно перезапустить командой `docker start`, но если они находятся в остановленном состоянии довольно долго, то удаляются
`docker start` *имя*	Запуск всех остановленных контейнеров с указанными *именами* или идентификаторами
`docker restart` *имя*	То же самое, что и команда `docker stop`, после которой автоматически выполняется команда `docker start`
`docker pause` *имя*	Приостановка всех процессов в контейнерах с указанными *именами* или идентификаторами. Сам контейнер продолжает работать

`docker unpause` *имя*	Возобновление работы контейнеров с указанными *именами* или идентификаторами
`docker kill` *имя*	Удаление всех контейнеров с указанными *именами* или идентификаторами

Управление контейнерами и образами

`docker cp` *путь1 путь2*	Копирование файлов в контейнер и из контейнера. *путь1* — это путь в вашей локальной файловой системе, а *путь2* — в контейнере. Путь в контейнере состоит из имени или идентификатора контейнера, двоеточия и пути к файлу, например `45171c25a5a0:/etc`
`docker exec` *имя команда*	Выполнение *команды* внутри контейнера с указанным *именем*, например `docker exec dcf10812030b ls -l`
`docker diff` *имя*	Перечисление удалений и изменений файлов в контейнере с момента его запуска
`docker rename` *старое_имя новое_имя*	Переименование контейнера
`docker rm [-f]` *имя*	Удаление контейнера. Используйте `-f` для принудительного удаления
`docker rmi [-f]` *образ*	Удаление образа. Задействуйте `-f` для принудительного удаления

Мониторинг контейнеров

`docker ps [-a]`	Перечисление запущенных контейнеров. Добавьте `-a`, чтобы вывести список всех контейнеров
`docker top` *имя*	Перечисление процессов, запущенных в контейнере с указанным *именем*
`docker logs [`*параметр(ы)*`]` *имя*	Просмотр системного журнала контейнера с указанным *именем*. Параметр `--follow` выводит журналы и продолжает их отслеживать. Параметр `--tail` *N* печатает последние *N* строк журнала. Параметр `--since` *T* начинает запись журналов с метки времени *T*, например `-c 2024-01-31`

Я рассмотрел только основы работы с командой `docker`. Чтобы узнать больше, обратитесь к документации (https://oreil.ly/6eNA6).

Просмотр и обработка изображений

`display`	Просмотр графического файла
`convert`	Преобразование файлов из одного графического формата в другой
`mogrify`	Изменение графического файла
`montage`	Объединение графических файлов

Для просмотра и редактирования изображений есть удобные инструменты с массой возможностей. Мы остановимся на инструментах командной строки из пакета ImageMagick (https://oreil.ly/HyBKC). Все его команды имеют схожее применение, а полное объяснение вы найдете на сайте https://oreil.ly/nJZnm.

display

stdin stdout **-file** **--opt** **--help** **--version**

```
display [параметр(ы)] файл
```

Команда `display` отображает изображения различных форматов: JPEG, PNG, GIF, BMP и др. Кроме того, команда включает в себя небольшой набор инструментов для редактирования изображений, которые появляются при щелчке левой кнопкой мыши на отображаемом изображении. Введите `q`, чтобы выйти из программы:

```
→ display photo.jpg
```

Это очень мощная команда — если вы откроете документацию, то увидите более 100 параметров.

Полезные параметры

`-resize` *размер*	Изменение размера изображения. Значения *размера* очень гибкие, включают установку ширины (`800`), высоты (`x600`), обоих значений (`800x600`), процентов для увеличения или уменьшения (`50%`), области в пикселях (`480000@`) и т. д.
`-flip`	Поворот изображения по вертикали
`-flop`	Поворот изображения по горизонтали
`-rotate` *N*	Поворот изображения на *N* градусов

`-backdrop`	Вывод изображения на однотонном заднике
`-fill`	Определение цвета задника, задаваемого параметром `-backdrop`
`-delay` *N*	Отображение изображения в течение *N* секунд. Если перечислены несколько изображений, то они выводятся в режиме слайд-шоу с задержкой в *N* секунд
`-identify`	Передача информации о формате, размере и других характеристиках изображения на стандартный вывод

convert

stdin stdout -file --opt --help --version

`convert [`*параметр(ы)*`] `*источник*` [`*параметр(ы)*`] `*назначение*

Команда `convert` копирует изображение и преобразует его в другой графический формат. Например, если у вас есть файл JPEG, можете создать файл PNG или PDF с тем же изображением:

```
→ convert photo.jpg newphoto.png
→ convert photo.jpg newphoto.pdf
```

Одновременно вы можете изменить параметры копии, например ее размер или поворот:

```
→ convert photo.jpg -resize 50% -flip newphoto.png
```

Команда `convert` принимает практически те же параметры, что и `display`.

mogrify

stdin stdout -file --opt --help --version

`mogrify [`*параметр(ы)*`] `*файл*

Команда `mogrify` преобразует изображение так же, как и команда `convert`, но при этом изменяется исходный *файл* изображения, а не копия. (Именно поэтому `convert` — более безопасная команда для экспериментов с любимой фотографией.) Она принимает практически те же параметры, что и `display`:

```
→ mogrify -resize 25% photo.jpg
```

montage

stdin stdout -file --opt --help --version

`montage` *источник* [*параметр(ы)*] *назначение*

Команда `montage` создает один файл изображения из коллекции указанных файлов. Например, вы можете создать лист миниатюр в виде одного изображения, пометив каждую миниатюру оригинальным именем файла:

```
→ montage photo.jpg photo2.png photo3.gif \
  -geometry 120x176+10+10 -label '%f' outfile.jpg
```

Команда `montage` позволяет настраивать результат. Так, показанная ранее команда создает миниатюры размером 120 × 176 пикселей, смещенные на 10 пикселей по горизонтали и вертикали (так появляется пространство между миниатюрами) и помеченные именем входного файла.

Полезные параметры

`-geometry` *ШхВ*[+|-]*Х*[+|-]*У*	Ширина (*Ш*), высота (*В*) и смещение (*Х*, *У*) изображения. Пример: `120x176+10-10`
`-frame` *N*	Создание рамки из *N* пикселей вокруг каждого изображения
`-label` *строка*	Пометка каждого изображения указанной *строкой*, которая может содержать специальные символы, начинающиеся со знака процента: `%f` — имя исходного файла, `%h` и `%w` — высота и ширина, `%m` — формат файла и около 40 других

Аудио и видео

`mediainfo`	Вывод подробной информации о мультимедийном файле
`cdparanoia`	Захват аудиодорожек с CD в WAV-файлы
`lame`	Преобразование WAV-файлов в MP3
`id3info`	Просмотр ID3-тегов в MP3-файле
`id3tag`	Редактирование ID3-тегов в MP3-файле
`ogginfo`	Просмотр информации о файле OGG
`metaflac`	Просмотр и изменение информации о файле FLAC
`sox`	Преобразование аудиофайлов из одного формата в другой

`mplayer`	Проигрывание видео- и аудиофайлов
`ffmpeg`	Преобразование видео- и аудиофайлов из одного формата в другой

Существует множество Linux-программ с графическими интерфейсами для воспроизведения и редактирования аудио- и видеофайлов, но мы по традиции рассмотрим инструменты командной строки.

mediainfo

stdin **stdout** **-file** --opt **--help** **--version**

```
mediainfo [параметр(ы)] файл(ы)
```

Команда `mediainfo` отображает подробную информацию о видео- и аудиофайлах.

```
→ mediainfo clip.mp4
General
Complete name                : clip.mp4
Format                       : MPEG-4
Format profile               : Base Media / Version 2
Codec ID                     : mp42 (mp42/mp41/isom/avc1)
File size                    : 1.11 MiB
Duration                     : 23 s 275 ms
Overall bit rate             : 399 kb/s
⋮
```

Команда `mediainfo` поддерживает параметры, но обычно достаточно вывода по умолчанию. Если вам нужен расширенный вывод, добавьте параметр `-f` (full), чтобы посмотреть все данные о медиафайле.

cdparanoia

stdin stdout -file **--opt** **--help** **--version**

```
cdparanoia [параметр(ы)] дорожка [назначение]
```

Команда `cdparanoia` захватывает (копирует) аудиоданные с CD и сохраняет их в файлы WAV (или других форматов, см. документацию). Обычно эта команда используется в следующих случаях.

`cdparanoia` *N*

Захват дорожки *N* в файл.

`cdparanoia -B`

Захват всех дорожек, имеющихся на CD, в отдельные файлы.

`cdparanoia -B 2-4`

Захват дорожек 2, 3 и 4 в отдельные файлы.

`cdparanoia 2-4`

Захват дорожек 2, 3 и 4 в один файл.

Если у вас возникли проблемы с доступом, попробуйте запустить `cdparanoia -Qvs` (что означает поиск приводов CD-ROM с подробным отчетом) и проанализировать отчеты программы.

lame

stdin stdout **-file** --opt **--help** **--version**

`lame [`*параметр(ы)*`] `*файл*`.wav`

Команда `lame` преобразует аудиофайл формата WAV (например, `song.wav`) в файл MP3:

```
→ lame song.wav song2.mp3
```

Команда поддерживает более 100 параметров, позволяющих управлять битрейтом, конвертировать аудиофайлы в другие форматы, добавлять ID3-теги и делать многое другое.

id3info

stdin stdout -file --opt **--help** **--version**

`id3info [`*параметр(ы)*`] [`*файл(ы)*`]`

Команда `id3info` выводит информацию об аудиофайле MP3, например название песни, имя исполнителя, название альбома и год выпуска. Файл должен содержать ID3-теги. Команда не поддерживает параметры, кроме `--help` и `--version`:

```
→ id3info knots.mp3
*** Tag information for knots.mp3
=== TIT2 (Title/songname/content description): Knots
=== TPE1 (Lead performer(s)/Soloist(s)): Gentle Giant
=== TALB (Album/Movie/Show title): Octopus
=== TYER (Year): 1972
*** mp3 info
MPEG1/layer III
Bitrate: 320KBps
Frequency: 44KHz
```

id3tag

stdin stdout **-file** **--opt** **--help** **--version**

`id3tag [`*`параметр(ы)`*`] `*`файл(ы)`*

Команда `id3tag` добавляет или изменяет ID3-теги в MP3-файле. Так, чтобы изменить название и исполнителя MP3-файла, выполните команду

```
→ id3tag -A "My Album" -a "Loud Linux Squad" song.mp3
```

Полезные параметры

`-A` *имя*	Установка *имени* исполнителя
`-a` *название*	Установка *названия* альбома
`-s` *название*	Установка *названия* дорожки
`-y` *год*	Установка *года*
`-t` *номер*	Установка *номера* дорожки
`-g` *номер*	Установка *номера* жанра

ogginfo

stdin stdout -file **--opt** --help --version

`ogginfo [`*`параметр(ы)`*`] [`*`файл(ы)`*`]`

`ogginfo` — это простая команда, отображающая информацию об аудиофайлах формата Ogg Vorbis:

```
→ ogginfo knots.ogg
Processing file "knots.ogg"...
⋮
User comments section follows...
        Title=Knots
        Artist=Gentle Giant
```

```
        Album=Octopus
        ⋮
Vorbis stream 1:
        Total data length: 69665 bytes
        Playback length: 0m:05.067s
        Average bitrate: 109.973744
```

Добавьте параметр `-h` для получения подробной информации о команде.

---

metaflac

stdin stdout -file **--opt** **--help** **--version**

```
metaflac [параметр(ы)] [файл(ы)]
```

Команда `metaflac` отображает или изменяет информацию об аудиофайле FLAC. Чтобы отобразить информацию, выполните команду

```
→ metaflac --list knots.flac
⋮
  sample_rate: 44100 Hz
  channels: 2
  bits-per-sample: 16
  total samples: 223488
⋮
  comments: 4
    comment[0]: Title=Knots
    comment[1]: Artist=Gentle Giant
    comment[2]: Album=Octopus
    comment[3]: Year=1972
```

Самый простой способ изменить информацию, например название и исполнителя, — это экспортировать ее в текстовый файл, отредактировать, а затем снова импортировать в аудиофайл:

```
→ metaflac --export-tags-to info.txt knots.flac
→ cat info.txt
Title=Knots
Artist=Gentle Giant
Album=Octopus
Year=1972
→ nano info.txt                    Внесите изменения и сохраните файл
→ metaflac --import-tags-from info.txt knots.flac
```

Полезные параметры

`--show-tag` *мег*	Вывод значения именованного *мега*, например исполнителя, названия альбома, года и т. д. Существует множество других параметров `show` (для получения подробной информации обратитесь к документации)
`--remove-tag` *мег*	Удаление из файла FLAC всех вхождений именованного *мега* (название, исполнитель и т. д.)

---

sox

stdin stdout -file --opt --help --version

```
sox [параметр(ы)] источник назначение
```

`sox` — это простейшая команда для конвертирования файлов из одного формата в другой. Она поддерживает форматы MP3, OGG, FLAC, WAV и десятки других (список можно увидеть с помощью команды `man soxformat`). Укажите новый формат, используя правильное расширение файла, как в этих примерах:

```
→ sox knots.mp3 knots2.wav      MP3 в WAV
→ sox knots.ogg knots2.mp3      OGG в MP3
→ sox knots.flac knots2.ogg     FLAC в OGG
```

У `sox` есть *множество* других возможностей, включая объединение аудиофайлов и добавление специальных эффектов. Подробности вы найдете в документации.

Полезные параметры

`-S`	Вывод индикатора выполнения, полезно при длительных преобразованиях
`--no-clobber`	Без перезаписи выходного файла, если он уже существует
`-t` *тип*	Указание *типа* входного файла, если `sox` не может его определить. Список типов приводится в `man soxformat`

mplayer

stdin stdout **-file** **--opt** **--help** **--version**

```
mplayer [параметр(ы)] видеофайл(ы)
```

Команда `mplayer` воспроизводит видео- и аудиофайлы во многих форматах (MPEG, AVI, MOV и др.):

```
→ mplayer clip.mp4
```

Чтобы приостановить и возобновить просмотр видео, нажмите клавишу Пробел, клавиши → и ← используются для перехода вперед и назад соответственно, а Q — для выхода из программы. Команда `mplayer` также воспроизводит аудиофайлы и поддерживает десятки параметров (подробности в документации). Больше информации есть на сайте https://oreil.ly/96S5B.

Среди других популярных видеоплееров для Linux — `vlc` (https://oreil.ly/hBGP0), `kaffeine` (https://oreil.ly/mROWt) и `xine` (https://oreil.ly/IyQw6).

ffmpeg

stdin stdout **-file** **--opt** **--help** **--version**

```
ffmpeg [параметр(ы)_источника] -i источник [параметр(ы)_
назначения] назначение
```

Команда `ffmpeg` конвертирует форматы видеофайлов, позволяет монтировать их, извлекает звуковые дорожки, создает постеры для видеороликов и делает многое другое. Чтобы досконально разобраться в работе команды, необходимо знать такие термины, как «частота кадров», «демультиплексирование» и «кодеки», но в этой книге я рассматриваю лишь несколько распространенных вариантов ее использования. Если вы выполняете сложную задачу, то рекомендую найти в интернете готовое решение для программы `ffmpeg`.

ПРИМЕЧАНИЕ

В отличие от большинства других команд Linux, команда ffmpeg учитывает порядок следования параметров. Куда бы вы ни поместили параметр -i *источник* в командной строке, параметры источника будут использоваться первыми, а параметры назначения — вторыми. Если параметр источника указан среди параметров назначения или наоборот, ffmpeg может завершить работу с ошибкой.

Конвертируем видеофайл `myvideo.mp4` из формата MP4 в формат MOV:

```
→ ffmpeg -i myvideo.mp4 myvideo.mov
```

Извлечем 10 секунд видео (`-t 10`), начиная с двухминутной отметки (`-s 00:02:00`), и сохраним их в файле `extract.mp4`:

```
→ ffmpeg -i myvideo.mp4 -ss 00:02:00 -t 10 -codec copy \
  extract.mp4
```

Извлечем звук из файла `myvideo.mp4` в MP3-файл `audio.mp3`:

```
→ ffmpeg -i myvideo.mp4 -q:a 0 -map a audio.mp3
```

Смонтируем несколько видеороликов, чтобы создать файл `movie.mp4`. Начнем с текстового файла, в котором указаны пути к исходным видеофайлам, а затем запустим команду `ffmpeg`:

```
→ cat videos.txt
file 'video1.mp4'
file 'video2.mp4'
file 'video3.mp4'
→ ffmpeg -f concat -safe 0 -i videos.txt -c copy movie.mp4
```

Создадим постер из видеофайла `myvideo.mp4`, извлекая один кадр (`-vframes 1`) на отметке в 5 секунд (`ss 5`) в файл JPEG `thumb.jpg` с разрешением 160 × 120 пикселей:

```
→ ffmpeg -ss 5 -i myvideo.mp4 -vcodec mjpeg \
  -vframes 1 -an -f rawvideo -s 160x120 thumb.jpg
```

В команде `ffmpeg` предусмотрена обширная справочная система (см. документацию):

```
→ ffmpeg -encoders                    Вывод списка поддерживаемых кодировщиков
→ ffmpeg --help encoder=mpeg4         Вывод подробной информации о кодировщике
```

Программирование с помощью сценариев командной оболочки

В bash встроен язык программирования. Вы можете писать *сценарии оболочки* (bash-программы) для выполнения задач, которые не позволяет решить ни одна команда. Команда `reset-lpg`, представленная в каталоге примеров книги, — это сценарий оболочки, который вы можете прочитать:

```
→ less ~/linuxpocketguide/reset-lpg
```

Как и в любом толковом языке программирования, в bash есть переменные, условные конструкции (`if-then-else`), циклы, ввод и вывод и многое другое. О сценариях оболочки написано много книг и ведутся онлайн-курсы, я же рассказываю о самом необходимом минимуме, чтобы вы могли начать практиковаться. Для получения более подробной информации выполните команду `info bash`, возьмите книгу по сценариям bash или найдите в интернете посвященные им учебники.

Создание и запуск сценариев оболочки

Чтобы создать сценарий оболочки, просто поместите команды bash в файл так, как будто вы набираете их в командной строке. Запустить сценарий можно тремя способами.

Добавить `#!/bin/bash` и сделать файл исполняемым

Это самый распространенный способ запуска сценариев. Добавьте строку

```
#!/bin/bash
```

в самый верх файла сценария. Это должна быть первая строка файла, выровненная по левому краю. Затем сделайте файл исполняемым:

```
→ chmod +x myscript
```

Запустите сценарий. Для сценария, находящегося в текущем каталоге, вам нужно добавить `./`, чтобы оболочка смогла его найти:

```
→ ./myscript
```

(Текущий каталог обычно не входит в путь поиска по соображениям безопасности. Не думаю, что вы хотите, чтобы вредоносный сценарий с именем ls в текущем каталоге замещал настоящую команду `ls`.)

В качестве альтернативы переместите сценарий в каталог в пути поиска и запустите его как любую другую команду:

```
→ myscript
```

Передать в bash

Запустите bash с именем файла сценария в качестве аргумента:

```
→ bash myscript
```

Запустить в текущей оболочке с командой `source` или точкой

Если вы хотите, чтобы сценарий внес изменения в текущую оболочку[1] (установил переменные, изменил каталог и т. д.), запустите его с помощью команды `source` или точки:

```
→ source myscript
→ . myscript
```

[1] Это происходит потому, что сценарий запускается в отдельной оболочке (дочерней), которая не может изменять исходную.

Пробельные символы и переносы строк

Сценарии оболочки Bash чувствительны к пробельным символам и переносам строк. Поскольку ключевые слова этого языка программирования на самом деле являются командами, оцениваемыми оболочкой, вы должны разделять аргументы пробельными символами. Аналогично, перенос строки сообщает оболочке, что команда завершена, поэтому символ переноса строки в середине команды может прервать ее выполнение. Следуйте примерам, которые я привожу здесь, и все будет в порядке. (Также смотрите мои советы по форматированию во врезке «Оформление сценариев оболочки» далее в этой главе.)

Чтобы разбить длинную команду на несколько строк, завершите каждую строку, кроме последней, символом \ (означает «продолжение на следующей строке»). Далее показан сценарий с длинной командой `grep`:

```
#!/bin/bash
grep abcdefghijklmnopqrstuvwxyz file1 file2 file3 \
file4 file5
```

Переменные

Я описал переменные оболочки в одноименном разделе главы 1:

```
→ MYVAR=6
→ echo $MYVAR
6
```

Все значения, хранящиеся в переменных, являются строками, но при необходимости оболочка обрабатывает числа корректно:

```
→ NUMBER="10"
→ expr $NUMBER + 5
15
```

Когда вы ссылаетесь на значение переменной в сценарии командной оболочки, рекомендуется заключать его в двойные кавычки, чтобы предотвратить ошибки во время

выполнения. Неопределенная переменная или переменная с пробелами в значении, если оно не окружено кавычками, будет оцениваться как нечто неожиданное, что приведет к сбою в работе сценария:

```
→ FILENAME="My Document"                    Пробел в имени
→ ls $FILENAME                              Попытка вывести список
ls: My: No such file or directory           Обнаружены два аргумента
ls: Document: No such file or directory
→ ls -l "$FILENAME"                         Перечисление в правильном порядке
My Document                                 Обнаружен один аргумент
```

Если имя переменной оценивается рядом с другой строкой, заключите его в фигурные скобки, чтобы предотвратить неожиданное поведение:

```
→ HAT="fedora"
→ echo "The plural of $HAT is $HATs"
The plural of fedora is                     Не существует переменной HATs
→ echo "The plural of $HAT is ${HAT}s"
The plural of fedora is fedoras             Желаемый результат
```

Ввод и вывод

Сценарии могут выводить данные на стандартный вывод с помощью команд `echo` и `printf`, которые я описал в разделе «Вывод на экран» данной главы:

```
→ echo "Hello world"
Hello world
→ printf "I am %d years old\n" `expr 20 + 20`
I am 40 years old
```

Сценарии могут считывать данные со стандартного ввода с помощью команды `read`, которая захватывает одну строку ввода и сохраняет ее в переменной:

```
→ read name
Sandy Smith <ENTER>
→ echo "I read the name $name"
I read the name Sandy Smith
```

См. также раздел «Аргументы командной строки» далее.

Логические переменные и коды выхода

Прежде чем я опишу условия и циклы, вам понадобится освоить понятие логического тестирования (истина/ложь). Для оболочки значение `0` означает истину или успех, а все остальное — ложь или неудачу. (Ноль — это «нет ошибки», а другие значения — коды ошибок.)

Кроме того, каждая команда Linux возвращает целочисленное значение в оболочку при выходе из команды. Это значение называется *кодом выхода*, *значением выхода* или *статусом выхода*. Вы можете увидеть его в специальной переменной `$?`:

```
→ cat exittest
My name is Sandy Smith and
I really like Ubuntu Linux
→ grep Smith exittest
My name is Sandy Smith and          Обнаружено совпадение…
→ echo $?
0…                                  поэтому код выхода — успех
→ grep aardvark exittest
→ echo $?                           Совпадение не обнаружено…
1 ..                                код выхода — неудача
```

Коды выхода команды описаны в документации к команде.

Команда test

Команда `test` (встроена в оболочку) оценивает простые логические выражения с числами и строками и устанавливает статус выхода `0` (true) или `1` (false):

```
→ test 10 -lt 5                     10 меньше 5?
→ echo $?
1                                   Это не так
→ test -n "hello"                   У hello ненулевая длина?
→ echo $?
0                                   Это так
```

Далее приведены общие аргументы команды `test` для проверки свойств целых чисел, строк и файлов.

Файловые тесты

`-d` *имя*	Файл *имя* — это каталог
`-f` *имя*	Файл *имя* — это обычный файл
`-L` *имя*	Файл *имя* — это символическая ссылка
`-r` *имя*	Файл *имя* доступен для чтения
`-w` *имя*	Файл *имя* доступен для записи
`-x` *имя*	Файл *имя* — исполняемый
`-s` *имя*	Файл *имя* имеет ненулевой размер
f1 `-nt` *f2*	Файл *f1* новее файла *f2*
f1 `-ot` *f2*	Файл *f1* старше файла *f2*

Строковые тесты

s1 `=` *s2*	Строка *s1* равна строке *s2*
s1 `!=` *s2*	Строка *s1* не равна строке *s2*
`-z` *s1*	Строка *s1* имеет нулевую длину
`-n` *s1*	Строка *s1* имеет ненулевую длину

Числовые тесты

a `-eq` *b*	Число *a* и число *b* равны
a `-ne` *b*	Число *a* и число *b* не равны
a `-gt` *b*	Число *a* больше числа *b*
a `-ge` *b*	Число *a* больше или равно числу *b*
a `-lt` *b*	Число *a* меньше числа *b*
a `-le` *b*	Число *a* меньше или равно числу *b*

Комбинированные тесты и отрицающие тесты

t1 `-a` *t2*	И — оба условия, *t1* и *t2*, истинны
t1 `-o` *t2*	ИЛИ — истинно либо условие *t1*, либо условие *t2*
`!` *условие*	Тест отрицания, то есть *условие* ложно
`\(` *условие* `\)`	Круглые скобки необходимы для группировки (как в алгебре)

Вы можете писать тесты в bash тремя способами. Первый — использовать команду `test`, как я показывал ранее. Второй

заключается в том, чтобы окружить условие двойными квадратными скобками:

```
→ [[ 10 -lt 5 ]]                    10 меньше 5?
→ echo $?
1                                   Это не так
→ [[ -n "hello" ]]                  У hello ненулевая длина?
→ echo $?
0                                   Это так
```

Третий способ, который поддерживается и некоторыми другими оболочками, заключается в использовании одной квадратной скобки:

```
→ [ 10 -lt 5 ]                      10 меньше 5?
→ echo $?
1                                   Это не так
→ [ -n "hello" ]                    У hello ненулевая длина?
→ echo $?
0                                   Это так
```

Одиночная квадратная скобка — это старый синтаксис с некоторыми странностями. Вы *должны* добавить пробельный символ после левой скобки и перед правой скобкой. Так нужно сделать потому, что левая скобка на самом деле является *командой с именем* `[` — это псевдоним для `test`. Поэтому за левой скобкой должны следовать *отдельные аргументы*, *разделенные пробельными символами*. Вы также должны убедиться, что последний аргумент — это правая квадратная скобка, означающая конец теста. Если вы по ошибке забудете про пробельные символы:

```
→ [ 5 -lt 4]                        Нет пробела между 4 и ]
bash: [: missing ']'
```

то `test` посчитает последним аргументом строку `4]` и обнаружит, что закрывающая скобка отсутствует.

Условные конструкции

Оператор `if` выбирает между альтернативами, каждая из которых может иметь сложную проверку. Простейшей формой является оператор `if-then`:

```
if команда                                    Если статус выхода равен 0
then
  тело
fi
```

Оформление сценариев оболочки

Ключевые слова в сценариях оболочки (`if`, `then`, `fi` и т. д.) должны стоять первыми в строке. Это означает, что ключевое слово должно следовать либо за символом перевода строки, либо за точкой с запятой (плюс необязательный пробельный символ). Существует еще два распространенных варианта оформления оператора `if`. Другие условия и циклы могут быть оформлены аналогичным образом:

```
if команда; then                    Точка с запятой перед then
  тело
fi

if команда; then тело; fi           Точка с запятой перед then и fi
```

Далее показан пример сценария с оператором `if`. Попробуйте запустить его с `sudo` и без него и посмотрите результаты:

```
→ cat script-if
#!/bin/bash
if [ `whoami` = "root" ]
then
  echo "You are the superuser"
fi
→ ./script-if                                 Нет вывода
→ sudo ./script-if
[sudo] password: xxxxxxxx
You are the superuser
```

Далее следует оператор `if-then-else`:

```
if команда
then
  тело1
else
  тело2
fi
```

Например:

```
→ cat script-else
#!/bin/bash
if [ `whoami` = "root" ]
then
  echo "You are the superuser"
else
  echo "You are a mere mortal"
fi
→ ./script-else
You are a mere mortal
→ sudo ./script-else
[sudo] password: xxxxxxxx
You are the superuser
```

Наконец, есть форма `if-then-elif-else`, в которой может быть сколько угодно тестов:

```
if команда1
then
  тело1
elif команда2
then
  тело2
elif ...
⋮
else
  телоN
fi
```

Например:

```
→ cat script-elif
#!/bin/bash
bribe=20000
if [ `whoami` = "root" ]
then
  echo "You are the superuser"
elif [ "$USER" = "root" ]
then
  echo "You might be the superuser"
elif [ "$bribe" -gt 10000 ]
then
  echo "You can pay to be the superuser"
else
```

```
  echo "You are still a mere mortal"
fi
→ ./script-elif
You can pay to be the superuser
```

Оператор `case` оценивает одно значение и переходит к соответствующему фрагменту кода:

```
→ cat script-case
#!/bin/bash
echo -n "What would you like to do (eat, sleep)? "
read answer
case "$answer" in
  eat)
    echo "OK, have a hamburger."
    ;;
  sleep)
    echo "Good night then."
    ;;
  *)
    echo "I'm not sure what you want to do."
    echo "I guess I'll see you tomorrow."
    ;;
esac
→ ./script-case
What would you like to do (eat, sleep)? sleep
Good night then.
```

Общая форма такова:

```
case значение in
  выражение1)
    тело1
    ;;
  выражение2)
    тело2
    ;;
  ⋮
  выражениеN)
    телоN
    ;;
  *)
    тело_конструкции_else
    ;;
esac
```

Указывается любое *значение*, обычно переменной, например `$myvar`, а *`выражение1`* и *`выражениеN`* — это шаблоны (для получения подробной информации выполните команду `info bash`) с заключительной `*` в роли `else`. Каждое *`тело`* должно завершаться символами `;;`, как показано в примере:

```
→ cat script-case2
#!/bin/bash
echo -n "Enter a letter: "
read letter
case $letter in
  X)
    echo "$letter is an X"
    ;;
  [aeiou])
    echo "$letter is a vowel"
    ;;
  [0-9])
    echo "$letter is a digit, silly"
    ;;
  *)
    echo "The letter '$letter' is not supported"
    ;;
esac
./script-case2
Enter a letter: e
e is a vowel
```

Циклы

Цикл `while` повторяет набор команд до тех пор, пока условие истинно:

```
while команда                    Пока статус выхода команды — 0
do
  тело
done
```

Например:

```
→ cat script-while
#!/bin/bash
i=0
while [ $i -lt 3 ]
do
  echo "$i"
```

```
  i=`expr $i + 1`
done
→ ./script-while
0
1
2
```

Цикл `until` повторяется до тех пор, пока условие не станет истинным:

```
until команда                    Пока статус выхода ненулевой
do
  тело
done
```

Например:

```
→ cat script-until
#!/bin/bash
i=0
until [ $i -ge 3 ]
do
  echo "$i"
  i=`expr $i + 1`
done
→ ./script-until
0
1
2
```

Остерегайтесь бесконечных циклов с `while` с условием, которое всегда оценивается как 0 (true), или циклов `until` с условием, которое всегда оценивается как ненулевое значение (false):

```
i=1
while [ $i -lt 10 ]
do
  echo "forever"               Ой, переменная i никогда не меняется
done                         Бесконечный цикл
```

Другой тип цикла, `for`, перебирает значения из списка:

```
for значение in список
do
  тело
done
```

Например:

```
→ cat script-for
#!/bin/bash
for name in Tom Jane Harry
do
  echo "$name is my friend"
done
→ ./script-for
Tom is my friend
Jane is my friend
Harry is my friend
```

Цикл `for` удобен для обработки списков файлов, например имен файлов с определенным расширением в текущем каталоге:

```
→ cat script-for2
#!/bin/bash
for file in *.docx
do
  echo "$file is a stinky Microsoft Word file"
done
→ ./script-for2
letter.docx is a stinky Microsoft Word file
shopping.docx is a stinky Microsoft Word file
```

Вы также можете создать список значений и зациклиться на них, применив фигурные скобки (см. раздел «Использование скобок» главы 1) или команду `seq` (см. раздел «Вывод на экран» данной главы):

```
→ cat script-seq
#!/bin/bash
for i in {1..20}                    Генерация целых чисел от 1 до 20
do
  echo "iteration $i"
done
→ ./script-seq
iteration 1
iteration 2
iteration 3
⋮
iteration 20
```

Аргументы командной строки

Сценарии оболочки могут принимать аргументы и параметры командной строки, как и другие команды Linux. (Честно говоря, многие команды Linux являются сценариями.) В сценарии оболочки эти аргументы обозначаются `$1`, `$2`, `$3` и т. д.:

```
→ cat script-args
#!/bin/bash
echo "My name is $1 and I come from $2"

→ ./script-args Johnson Wisconsin
My name is Johnson and I come from Wisconsin
→ ./script-args Bob
My name is Bob and I come from
```

Сценарий может проверить количество полученных аргументов с помощью `$#`:

```
→ cat script-args2
#!/bin/bash
if [ $# -lt 2 ]
then
  echo "$0 error: you must supply two arguments"
else
  echo "My name is $1 and I come from $2"
fi
```

Специальное значение `$0` содержит имя сценария и подходит для отправки сообщений об использовании и ошибках:

```
→ ./script-args2 Barbara
./script-args2 error: you must supply two arguments
```

Чтобы перебрать все аргументы командной строки, задействуйте цикл `for` со специальной переменной `$@`, в которой хранятся все аргументы:

```
→ cat script-args3
#!/bin/bash
for arg in $@
do
  echo "I found the argument $arg"
done
```

```
→ ./script-args3 One Two Three
I found the argument One
I found the argument Two
I found the argument Three
```

Завершение работы с помощью команды exit

Команда `exit` завершает работу сценария и передает оболочке указанный код выхода (см. раздел «Логические переменные и коды выхода» ранее в данной главе). Как вы уже узнали, сценарии должны возвращать 0 в случае успеха и 1 (или другое ненулевое значение) в случае неудачи. Если ваш сценарий не вызывает `exit`, его код выхода будет идентичен коду последней команды, которую выполнил сценарий:

```
→ cat script-exit
#!/bin/bash
if [ $# -lt 2 ]
then
  echo "$0 error: you must supply two arguments"
  exit 1
else
  echo "My name is $1 and I come from $2"
fi
exit 0
→ ./script-exit Bob
./script-exit error: you must supply two arguments
→ echo $?
1
```

Конвейеризация данных в bash

Bash — это не просто оболочка, есть еще и команда `bash`, которая считывает данные со стандартного ввода. Это значит, что вы можете конструировать команды в виде строк и отправлять их bash для выполнения:

```
→ echo wc -l myfile
wc -l myfile
→ echo wc -l myfile | bash
18 myfile
```

ВНИМАНИЕ!

Конвейеризация команд в bash — это мощный инструмент, который может быть и опасным. Во-первых, убедитесь, что вы точно знаете, какие команды передаете bash для выполнения. Вы же не хотите передать туда неожиданную команду rm и удалить нужный файл (или 1000 ценных файлов)?

Если кто-то попросит вас скачать текст из интернета (например, с помощью команды curl) и передать его bash, не делайте этого напрямую. Сначала захватите веб-страницу в виде файла (с помощью curl или wget), внимательно изучите ее и примите взвешенное решение, стоит обрабатывать ее с помощью bash или нет.

Эта техника невероятно полезна. Предположим, вы хотите загрузить с веб-сайта файлы `photo1.jpg`, `photo2.jpg`, `photo100.jpg`. Вместо того чтобы набирать вручную 100 команд `wget`, зациклите команду, используя `seq` для построения списка целых чисел от 1 до 100:

```
→ for i in `seq 1 100`
do
  echo wget https://example.com/photo$i.jpg
done
wget https://example.com/photo1.jpg
wget https://example.com/photo2.jpg
⋮
wget https://example.com/photo100.jpg
```

Да, вы создали текст из 100 команд. Теперь передайте вывод программе bash, которая выполнит их все:

```
→ for i in `seq 1 100`
do
  echo wget https://example.com/photo$i.jpg
done | bash
```

Вот более сложное, но практичное применение. Предположим, у вас есть набор файлов, которые вы хотите переименовать. Поместите старые имена в файл `oldnames`, а новые — в файл `newnames`:

```
→ cat oldnames
oldname1
oldname2
oldname3
→ cat newnames
newname1
newname2
newname3
```

Теперь с помощью команд `paste` и `sed` (см. раздел «Работа с текстом в файлах» главы 2) поместите старые и новые имена рядом, добавьте слово mv к каждой строке, и у вас получится последовательность команд `mv`:

```
→ cat oldnames \
  | paste -d' ' oldnames newnames \
  | sed 's/^/mv /'
mv oldfile1 newfile1
mv oldfile2 newfile2
mv oldfile3 newfile3
```

Наконец, передайте вывод bash, и смена имени будет завершена за мгновение:

```
→ cat oldnames \
  | paste -d' ' oldnames newnames \
  | sed 's/^/mv /' \
  | bash
```

За пределами оболочки…

Сценарии оболочки подходят для многих целей, но в Linux есть более мощные сценарные языки, а также скомпилированные языки программирования. Вот некоторые из них.

Язык	Программа	Открыть программу…
C, C++	gcc, g++	man gcc
		https://oreil.ly/-kuj1
Java	javac	https://oreil.ly/-yvQR
.NET	mono	man mono
		https://oreil.ly/IVenL

Язык	Программа	Открыть программу…
Perl	perl	`man perl`
		https://oreil.ly/LKgdM
PHP	php	`man php`
		https://oreil.ly/JPmIL
Python	python	`man python`
		https://oreil.ly/vAunc
Ruby	ruby	https://oreil.ly/ifm2L

Пара слов на прощание

Хотя я рассказал о многих командах и возможностях Linux, вам еще предстоит многое узнать. Надеюсь, вы продолжите изучать возможности своей системы Linux.

Чтобы расширить свои навыки работы с Linux, ознакомьтесь с моей следующей книгой *Efficient Linux at the Command Line*[1]. В ней вы найдете множество практических советов и методик, которые помогут быстрее и эффективнее работать с Linux. Для получения дополнительной информации посетите мой сайт https://danieljbarrett.com.

[1] https://learning.oreilly.com/library/view/efficient-linux-at/9781098113391/.

Об авторе

Дэниел Джей Барретт — преподаватель, пишет о Linux и на смежные темы уже более 30 лет. Среди его многочисленных книг — *Efficient Linux at the Command Line*, *Linux Pocket Guide*, *SSH, The Secure Shell: The Definitive Guide*, *Linux Security Cookbook*, *Macintosh Terminal Pocket Guide* и *MediaWiki*. У Дэна большой и разнообразный опыт: он был инженером-программистом, играл в рок-группе, работал системным администратором, преподавал в университете, трудился веб-дизайнером и шутил на публику. Сейчас Дэниел работает в компании Google. Посетите сайт DanielJBarrett.com, чтобы узнать о нем побольше.

Иллюстрация на обложке

Животное на обложке этой книги — *бельгийский тяжеловоз*, или *брабансон*. Первое упоминание этих нежных гигантов датируется XVII веком, а первая племенная книга была выпущена в 1886 году. Бельгийские тяжеловозы — потомки дестриэ, боевого друга рыцарей. Брабансоны были выведены для работы в промышленности, на фермах и для перевозки многотонных грузов. У лошадей спокойный, мягкий и бесстрашный нрав, им нравится работать, и ими легко управлять. К тому же брабансоны легко приспосабливаются к любым условиям.

У бельгийских тяжеловозов есть одна особенность: очень маленькая голова по сравнению с телом. У них прямой профиль и добрые глаза. Их отличает компактное коренастое тело с толстой шеей, массивной грудью и широкой спиной, особенно у жеребцов. Большие копыта закрыты густыми щетками. Их рост — от 170 до 180 см. Лошади растут до 4–5 лет, половозрелый конь в 5 лет может весить от 800 до 1000 кг.

Несмотря на то что предшественники брабансонов вымерли, бельгийские тяжеловозы — очень популярная порода тяжеловозов, которой не грозит вымирание.

Многие животные, изображенные на обложках книг O'Reilly, находятся под угрозой исчезновения, но все они очень важны для мира.

На обложке — иллюстрация Кейт Монтгомери, созданная по мотивам старинной линейной гравюры из книги Вуда *Animate Creation*.

Дэниел Джей Барретт

Linux. Карманный справочник

4-е издание

Перевел с английского С. Черников

Изготовлено в России. Изготовитель: ТОО «Спринт Бук».
Место нахождения и фактический адрес: 010000 Казахстан, город Астана,
район Алматы, Проспект Ракымжан Кошкарбаев, д. 10/1, н. п. 18.

Дата изготовления: 11.2024. Наименование: книжная продукция.
Срок годности: не ограничен.

Подписано в печать 03.10.24. Формат 84×108/32. Бумага офсетная. Усл. п. л. 16,800.
Тираж 1000. Заказ 0000