O'REILLY®

командная строка Linux

лучшие практики

Дэниел Джей Барретт

Efficient Linux at the Command Line

Boost Your Command-Line Skills

Daniel J. Barrett

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

Linux
Командная строка

Лучшие практики

Дэниел Джей Барретт

ПИТЕР®

Санкт-Петербург • Москва • Минск

2023

ББК 32.973.2-018.2
УДК 004.451
Б25

Барретт Дэниел Джей

Б25 Linux. Командная строка. Лучшие практики. — СПб.: Питер, 2023. — 256 с.: ил. — (Серия «Бестселлеры O'Reilly»).

ISBN 978-5-4461-2300-1

Перейдите на новый уровень работы в Linux! Если вы системный администратор, разработчик программного обеспечения, SRE-инженер или пользователь Linux, книга поможет вам работать быстрее, элегантнее и эффективнее. Вы научитесь создавать и запускать сложные команды, которые решают реальные бизнес-задачи, обрабатывать и извлекать информацию, а также автоматизировать ручную работу.

Узнайте, что происходит внутри командной оболочки Linux. Вне зависимости от используемых команд вы повысите эффективность работы в Linux и станете более конкурентоспособным специалистом.

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

ББК 32.973.2-018.2
УДК 004.451

ISBN 978-1098113407 англ.

ISBN 978-5-4461-2300-1

Оглавление

Предисловие

Эт книг позволит перейти н новый уровень использов ния ком ндной строки Linux, чтобы вы могли р бот ть быстрее, умнее и эффективнее.

Если вы похожи н большинство пользов телей Linux, то приобрели первон ч льные н выки использов ния ком ндной строки н р боте, или прочит в вводную книгу, или уст новив Linux дом и просто попробов в. Моя книг должн помочь в м сдел ть следующий ш г — р звить н выки р боты с ком ндной строкой Linux от среднего до продвинутого уровня. Он н полнен метод ми и концепциями, которые, я н деюсь, изменят в ше вз имодействие с Linux и повысят производительность. Восприним йте ее к к вторую книгу по использов нию Linux, котор я выведет в с з р мки зов.

Ком ндн я строк — с мый простой и вместе с тем с мый сложный из интерфейсов. Он прост, потому что содержит только пригл шение ком ндной строки, которое ждет в ших действий[1]:

```
$
```

Он сложный, потому что з все, что выходит з р мки пригл шения, отвеч ете только вы. Нет понятных зн чков, кнопок или меню, которые могли бы помочь. Вместо этого к жд я введенн я ком нд является результ том в ших усилий. Это верно к к для основных ком нд, т ких, н пример, к к вывод списк ф йлов:

```
$ ls
```

т к и для более сложных, н пример:

```
$ paste <(echo {1..10}.jpg | sed 's/ /\n/g') \
        <(echo {0..9}.jpg | sed 's/ /\n/g') \
  | sed 's/^/mv /' \
  | bash
```

Если вы смотрите н предыдущую ком нду и дум ете: «*Что это*, черт возьми?» или «Мне бы никогд не пон добил сь т к я сложн я ком нд », то эт книг для в с[2].

---

[1] Для отобр жения пригл шения ком ндной строки в этой книге используется символ $. В в шей опер ционной системе этот символ может быть другим.

[2] Вы узн ете н зн чение этой т инственной ком нды в гл ве 8.

Чему вы научитесь

Эт книг поможет в м быстро и эффективно освоить три основных н вык :

- Выбор и созд ние ком нд для решения текущей з д чи.
- Эффективный з пуск ком нд.
- Удобн я н виг ция по ф йловой системе Linux.

Прочит в книгу, вы будете поним ть, что ост ется з к дром при з пуске ком нды, и н учитесь лучше предск зыв ть ее результ ты (не руководствов ться суевериями). Вы позн комитесь с дюжиной р зных способов з пуск ком нд и узн ете, когд лучше использов ть к ждый из них. Вы т кже узн ете приемы и хитрости, которые сдел ют в шу р боту продуктивнее. Н пример:

- Построение сложных ком нд из простых, ш г з ш гом, для решения пр ктических з д ч, н пример упр вления п ролями или созд ния тысяч тестовых ф йлов.
- Экономия времени з счет продум нной орг низ ции дом шнего к т лог , чтобы в м не приходилось иск ть нужные ф йлы.
- Преобр зов ние текстовых ф йлов и получение из них д нных для решения пост вленных з д ч.
- Упр вление из ком ндной строки теми опер циями в Linux, которые обычно выполняются мышкой. Н пример, копиров ние и вст вк с помощью буфер обмен или получение и обр ботк веб-д нных. И все это, не отрыв я рук от кл ви туры.

И с мое гл вное, вы изучите лучшие пр ктические методы р боты в ком ндной строке, поэтому нез висимо от того, к кие ком нды вы з пуск ете, вы сможете добиться большего в повседневном использов нии Linux и ст ть более конкурентоспособными н рынке труд . Я бы хотел иметь эту книгу, когд с м только н чин л изучение Linux.

Чем эта книга не является

Эт книг не поможет оптимизиров ть и не сдел ет эффективнее в ш компьютер с уст новленной опер ционной системой Linux. Он лишь сдел ет *в с* более эффективными при использов нии Linux.

Эт книг т кже не является исчерпыв ющим спр вочником по ком ндной строке. Существуют сотни ком нд и функций, которые здесь не упомянуты. Эт книг об опыте. Он содержит тщ тельно отобр нный н бор пр ктических зн ний о ком ндной строке для р звития в ших н выков. В к честве спр вочного руководств используйте мою предыдущую книгу *Linux Pocket Guide* (изд тельство O'Reilly).

Для кого эта книга

Предпол г ется, что у в с есть опыт р боты с Linux, поскольку *эт книг не зн комит с Linux*. Он предн зн чен , прежде всего, для пользов телей, которые хотят улучшить свои н выки р боты с ком ндной строкой, т ких к к студенты, системные дминистр торы, р зр ботчики прогр ммного обеспечения, IT-инженеры, тестировщики и энтузи сты опер ционной системы Linux. Опытные пользов тели Linux т кже смогут н йти для себя полезный м тери л, особенно если они учились методом проб и ошибок и хотят улучшить свое пон ним ние ком ндной строки.

Чтобы извлечь м ксим льную пользу из этой книги, вы уже должны быть зн комы со следующими тем ми (если нет, см. приложение А, где д ется их кр ткий обзор):

- Созд ние и ред ктиров ние текстовых ф йлов с помощью ред кторов `vim (vi), emacs, nano, pico`.
- Основные ком нды для р боты с ф йл ми, т кие к к `cp` (копиров ние), `mv` (перемещение или переименов ние), `rm` (уд ление), `chmod` (изменение пр в доступ).
- Основные ком нды для просмотр ф лов, т кие к к `cat` (вывод содержимого ф йл) и `less` (постр ничный вывод текст).
- Основные ком нды для р боты с к т лог ми, т кие к к `cd` (переход в другой к т лог), `ls` (вывод списк ф йлов и к т логов), `mkdir` (созд ние к т лог), `rmdir` (уд ление к т лог), `pwd` (вывод пути к текущему к т логу).
- Основы сцен риев ком ндной оболочки: хр нение ком нд Linux в ф йле, созд ние исполняемого ф йл (с помощью `chmod 755` или `chmod +x`) и его з пуск.
- Просмотр встроенной документ ции Linux, известной к к man-стр ницы, с помощью ком нды `man` (н пример, `man cat` отобр ж ет документ цию по ком нде `cat`).
- Умение ст ть суперпользов телем с помощью ком нды `sudo` для полного доступ к опер ционной системе (н пример, ком нд `sudo nano /etc/hosts` позволяет ред ктиров ть ф йл */etc/hosts*, изменение которого з прещено обычному пользов телю).

Если вы т кже зн комы с общими функциями ком ндной строки, т кими к к сопост вление имен ф йлов с ш блоном (символы `*` и `?`), перен пр вление ввод /вывод (`<` и `>`) и к н лы (`|`), то вы уже хорошо подготовлены к изучению этой книги.

Ваша командная оболочка

Предпол г ется, что в шей ком ндной оболочкой является bash, котор я используется по умолч нию в большинстве дистрибутивов Linux. Под термином «оболочк » в тексте имеется в виду именно bash. Большинство идей, предст вленных в книге, применимы и к другим оболочк м, т ким к к zsh или dash (см. Приложение Б для перевод примеров в другие оболочки). Больш я ч сть м тери л будет р бот ть без изменений и в Apple Mac-термин ле, который по умолч нию з пуск ет zsh, но т кже может з пуск ть и bash[1].

Условные обозначения

В этой книге используются следующие условные обозн чения.

Курсив

Курсивом выделены новые термины и в жные понятия, т кже имен ф йлов и к т логов.

`Моноширинный шрифт`

Используется для листингов прогр мм, т кже внутри бз цев для обозн чения т ких элементов, к к переменные и функции, б зы д нных, типы д нных, переменные среды, опер торы и ключевые слов , имен ф йлов и их р сширения.

`Моноширинный полужирный шрифт`

Пок зыв ет ком нды или другой текст, который пользов тель должен ввести с мостоятельно.

`Моноширинный курсив`

Пок зыв ет текст, который должен быть з менен зн чениями, введенными пользов телем или определяемыми контекстом. Кроме того, этот шрифт используется для кр тких пояснений в пр вой ч сти листингов.

`Моноширинный шрифт в рамке`

Используется для привлечения вним ния к отдельным ч стям листингов.

Шрифт без засечек

Используется для обозн чения URL, дресов электронной почты, н зв ний кнопок и других элементов интерфейс .

---

[1] Версия bash для macOS уст рел , в ней отсутствуют в жные функции. Чтобы обновить bash, см. ст тью Дэниел Вейбеля (Daniel Weibel) *Upgrading Bash on macOS* (https://oreil.ly/35jux).

Этот рисунок ук зыв ет н совет или предложение.

Этот рисунок ук зыв ет н примеч ние.

Этот рисунок ук зыв ет н предупреждение.

Использование исходного кода примеров

Вспомог тельные м тери лы (примеры код , упр жнения и т. д.) доступны для з грузки по дресу: https://efficientlinux.com/examples. Если у в с возникнут вопросы технического х р ктер по использов нию примеров код , н пр вляйте их по электронной почте н дрес bookquestions@oreilly.com.

В общем случ е все примеры код из книги вы можете использов ть в своих прогр мм х и в документ ции. В м не нужно обр щ ться в изд тельство з р зрешением, если вы не собир етесь воспроизводить существенные ч сти прогр ммного код . Если вы р зр б тыв ете прогр мму и используете в ней несколько фр гментов код из книги, в м не нужно обр щ ться з р зрешением. Но для прод жи или р спростр нения примеров из книги в м потребуется р зрешение от изд тельств O'Reilly. Вы можете отвеч ть н вопросы, цитируя д нную книгу или примеры из нее, но для включения существенных объемов прогр ммного код из книги в документ цию в шего продукт потребуется р зрешение.

Мы рекомендуем, но не требуем доб влять ссылку н первоисточник при цитиров нии. Под ссылкой н первоисточник мы подр зумев ем ук з ние второв, изд тельств и ISBN.

З р зрешением н использов ние зн чительных объемов прогр ммного код из книги обр щ йтесь по дресу permissions@oreilly.com.

Благодарности

Эту книгу было приятно пис ть. Сп сибо з меч тельным людям из изд тельств O'Reilly, особенно ред ктор м Вирджинии Уилсон (Virginia Wilson) и Джону Девинсу (John Devins), ред ктор м производств Кейтлин Гег н (Caitlin Ghegan) и Грегори Х йм ну (Gregory Hyman), контент-менеджеру Кристен Бр ун (Kristen Brown), ред ктору Киму Уимпсетту (Kim Wimpsett), ред ктору предметного ук з теля Сью Клефст д (Sue Klefstad) и всегд готовой помочь ком нде производственного отдел . Я т кже очень бл год рен рецензент м книги Полу Б йеру (Paul Bayer), Джону Бонезио (John Bonesio), Дэну Риттеру (Dan Ritter) и К рле Шредер (Carla Schroder) з ценные коммент рии и критические з меч ния. Т кже сп сибо Boston Linux Users Group з предложения по н зв нию. Особ я бл год рность Мэгги Джонсон (Maggie Johnson) из Google з ее любезное р зрешение н пис ть книгу.

Я бы хотел выр зить глубоч йшую бл год рность Чипу Эндрюсу (Chip Andrews), Мэтью Ди зу (Matthew Diaz) и Роберту Стр нду (Robert Strandh), которые 35 лет н з д учились со мной в Университете Джонс Хопкинс (The Johns Hopkins University). Они з метили мой р стущий интерес к опер ционной системе Unix и, к моему кр йнему удивлению, порекомендов ли Деп рт менту компьютерных н ук (Computer Science Department) н нять меня в к честве системного дминистр тор . Их вер в меня изменил тр екторию моей жизни (кст ти, бл год ря Роберту доб влен совет к с тельно н выков слепой печ ти в гл ве 3). Сп сибо т кже тем, кто созд ет и поддержив ет проекты Linux, GNU Emacs, Git, AsciiDoc и многие другие инструменты с открытым исходным кодом — без этих умных и щедрых людей моя к рьер действительно был бы совсем другой.

К к всегд , сп сибо моей з меч тельной семье, Лизе и Софии, з их любовь и терпение.

От издательства

В ши з меч ния, предложения, вопросы отпр вляйте по дресу comp@piter.com (изд тельство «Питер», компьютерн я ред кция).

Мы будем р ды узн ть в ше мнение!

Н веб-с йте изд тельств www.piter.com вы н йдете подробную информ цию о н ших книг х.

ЧАСТЬ 1

Основные понятия

Первые четыре гл вы должны быстро повысить эффективность в шей р боты в ком ндной строке, позн комив в с с принцип ми и метод ми, которые можно ср зу применить н пр ктике. Вы н учитесь комбиниров ть ком нды с помощью к н лов, поймете функции ком ндной оболочки Linux, сможете повторно вызыв ть и ред ктиров ть ком нды, которые р нее уже з пуск ли, и быстро перемещ ться по ф йловой системе Linux.

ГЛАВА 1

Объединение команд

В Windows, macOS и большинстве других опер ционных систем вы тр тите время н з пуск т ких приложений, к к веб-бр узеры, текстовые ред кторы, прогр ммы для р боты с т блиц ми и игры. Типичное приложение содержит множество функций: все, что, по мнению р зр ботчиков, может пон добиться пользов телям. Поэтому большинство приложений с модост точны и не связ ны с другими приложениями. Вы можете время от времени копиров ть и вст влять д нные из одного приложения в другое, но по большей ч сти они нез висимы друг от друг .

Ком ндн я строк в Linux р бот ет по-другому. Вместо больших приложений с множеством функций Linux предост вляет тысячи небольших ком нд с огр ниченным н бором возможностей. Ком нд `cat`, н пример, выводит н экр н содержимое ф йлов и больше ничего. `ls` отобр ж ет список ф йлов в к т логе, `mv` переименовыв ет ф йлы и т. д. К жд я ком нд имеет простую, четко определенную цель.

Что дел ть, если в м нужно произвести более сложные действия? Не волнуйтесь, Linux позволяет легко *объединять ком нды*, чтобы их функции р бот ли вместе для достижения пост вленной цели. Т кой способ порожд ет совсем другое отношение к р боте з компьютером. Вместо того чтобы спр шив ть себя «К кое приложение мне з пустить?» для достижения необходимого результ т , возник ет вопрос: «К кие ком нды мне объединить?»

В этой гл ве вы узн ете, к к упорядочив ть и з пуск ть ком нды в р зличных комбин циях, чтобы реш ть пост вленные з д чи. Для простоты н чнем с шести ком нд Linux и их основных способов применения. Это позволит сосредоточиться н более сложной и интересной ч сти — их объединении — без долгого обучения. Это все р вно, что учиться готовить, используя только шесть ингредиентов, или осв ив ть столярное дело с применением только молотк и пилы (дополнительные ком нды доб вятся в в ш н бор в гл ве 5).

Вы будете объединять ком нды, используя *к н л* (*pipe*) — функцию Linux, котор я соединяет вывод одной ком нды с вводом другой. При предст влении к ждой ком нды (`wc`, `head`, `cut`, `grep`, `sort` и `uniq`) мы ср зу же будем р см трив ть пример использов ния к н л . Некоторые примеры будут полезны

для повседневной р боты в Linux, в то время к к другие являются искусственно выдум нными для демонстр ции в жной функции.

Ввод, вывод и каналы

Большинство ком нд Linux считыв ют ввод с кл ви туры, отобр ж ют вывод н экр не или дел ют и то, и другое. В Linux эти опер ции чтения и з писи имеют собственные н зв ния:

stdin (*ст нд ртный ввод,* или *ст нд ртный входной поток*)

> Поток д нных, который Linux считыв ет при вводе с кл ви туры. Когд вы вводите любую ком нду в термин ле после пригл шения ком ндной строки, вы перед ете д нные в ст нд ртный входной поток *stdin*.

stdout (*ст нд ртный вывод,* или *ст нд ртный выходной поток*)

> Поток д нных, который Linux выводит н в ш дисплей. Когд вы з пуск ете ком нду `ls` для печ ти имен ф йлов, результ ты появляются в ст нд ртном выходном потоке *stdout*.

Теперь с мое интересное. Вы можете подключить ст нд ртный вывод одной ком нды к ст нд ртному вводу другой, чтобы перв я ком нд перед в л свои выходные д нные во вторую. Д в йте н чнем со зн комой ком нды `ls -l` для просмотр большого к т лог , т кого к к */bin*, в длинном форм те:

```
$ ls -l /bin
total 12104
-rwxr-xr-x 1 root root 1113504 Jun  6  2019 bash
-rwxr-xr-x 1 root root  170456 Sep 21  2019 bsd-csh
-rwxr-xr-x 1 root root   34888 Jul  4  2019 bunzip2
-rwxr-xr-x 1 root root 2062296 Sep 18  2020 busybox
-rwxr-xr-x 1 root root   34888 Jul  4  2019 bzcat
⋮
-rwxr-xr-x 1 root root    5047 Apr 27  2017 znew
```

Этот к т лог содержит зн чительно больше ф йлов, чем, строк н в шем дисплее, поэтому вывод быстро прокручив ется з пределы экр н . Ж ль, что `ls` не может печ т ть информ цию по одному экр ну з р з, дел я п узу, пок вы не н жмете кл вишу для продолжения. Но постойте: у другой ком нды Linux есть т к я возможность. Ком нд `less` отобр ж ет ф йл по одному экр ну з р з:

```
$ less myfile
```

Вы можете соединить эти две ком нды, потому что `ls` перед ет д нные в ст нд ртный вывод, `less` может считыв ть д нные из ст нд ртного ввод . Используем к н л для отпр вки выходных д нных `ls` н вход ком нды `less`:

```
$ ls -l /bin | less
```

ЧТО ТАКОЕ КОМАНДА?

Слово *ком нд* имеет три р зличных зн чения в Linux, предст вленные н рис. 1.1:

Прогр мм
: Исполняем я прогр мм , н зв нн я и выполняем я одним словом, н пример `ls`, или н логичн я функция, встроенн я в оболочку, н пример `cd` (*встроенн я ком нд оболочки*)[1].

Прост я ком нд
: Имя прогр ммы или встроенной ком нды оболочки, з которым могут следов ть ргументы, н пример `ls -l /bin`.

Комбиниров нн я ком нд
: Несколько простых ком нд, р ссм трив емых к к единое целое, н пример конвейер `ls -l /bin | less`

Рис. 1.1. Программы, простые команды и комбинированные команды называются «командами»

В этой книге слово «ком нд » используется во всех этих смысл х. Обычно из контекст понятно, что имеется в виду, но если нет, будем использов ть один из более конкретных терминов.

Эт объединенн я ком нд отобр ж ет содержимое к т лог по одному экр ну з р з. Вертик льн я черт (|) между ком нд ми — это символ[2] к н л в Linux. Он соединяет ст нд ртный вывод первой ком нды со ст нд ртным вводом следующей. Ком ндн я строк , содерж щ я к н лы, н зыв ется *конвейером* (*pipeline*).

[1] Ст нд рт POSIX н зыв ет т кие ком нды *утилит ми*.

[2] Н ст нд ртной кл ви туре символ вертик льной черты н ходится н той же кл више, что и обр тн я кос я черт (\), обычно р сположенн я между кл виш ми Enter и Backspace или между левой кл вишей Shift и Z.

Ком нды обычно не зн ют, что они являются ч стью конвейер . `ls` счит ет, что выводит д нные н дисплей, хотя н с мом деле ее вывод был перен пр влен н `less`. А `less` верит, что чит ет д нные с кл ви туры, когд н с мом деле получ ет вывод `ls`.

Шесть команд для начала

Использов ние к н лов — неотъемлем я ч сть р боты с Linux. Д в йте р звив ть в ши н выки р боты с к н л ми с помощью небольшого н бор ком нд Linux. Когд вы столкнетесь с к кими-то из этих ком нд в будущем, то будете готовы их объединять.

Ком нды `wc, head, cut, grep, sort` и `uniq` имеют множество опций и режимов р боты, про которые пок умолчим, чтобы сосредоточиться н к н л х. Если хотите узн ть больше о конкретной ком нде, з пустите ком нду `man`, котор я отобр зит полную документ цию. Н пример:

```
$ man wc
```

Для демонстр ции этих шести ком нд в действии будем использов ть ф йл с именем *animals.txt*, в котором перечислены некоторые сведения из книг изд тельств O'Reilly. Содержимое ф йл предст влено в примере 1.1.

Пример 1.1. Содержимое файла animals.txt

```
python   Programming Python         2010  Lutz, Mark
snail    SSH, The Secure Shell      2005  Barrett, Daniel
alpaca   Intermediate Perl          2012  Schwartz, Randal
robin    MySQL High Availability    2014  Bell, Charles
horse    Linux in a Nutshell        2009  Siever, Ellen
donkey   Cisco IOS in a Nutshell    2005  Boney, James
oryx     Writing Word Macros        1999  Roman, Steven
```

К жд я строк содержит четыре ф кт о к кой-либо книге изд тельств O'Reilly, р зделенные одним символом т буляции: животное н обложке, н зв ние книги, год публик ции и имя первого втор .

Команда #1: wc

Ком нд `wc` выводит количество строк, слов и символов в ф йле:

```
$ wc animals.txt
  7 51 325 animals.txt
```

wc сообщ ет, что в ф йле *animals.txt* 7 строк, 51 слово и 325 символов. Если вы посчит ете символы, включ я пробелы и т буляции, то н йдете только 318 символов, но wc т кже учитыв ет скрытые символы новой строки.

Опции -l, -w и -c ук зыв ют wc печ т ть только количество строк, слов и символов соответственно:

```
$ wc -l animals.txt
7 animals.txt
$ wc -w animals.txt
51 animals.txt
$ wc -c animals.txt
325 animals.txt
```

Вывод п р метров текст — н столько полезн я и ч сто встреч ющ яся з д ч , что вторы wc р зр бот ли свою ком нду специ льно для р боты с к н л ми. Если вы не ук зыв ете имя ф йл , он чит ет д нные из ст нд ртного ввод и отпр вляет д нные в ст нд ртный вывод. Д в йте воспользуемся ls для вывод содержимого текущего к т лог и используем wc для подсчет строк. Этот конвейер отвеч ет н вопрос «Сколько ф йлов в моем текущем к т логе?»:

```
$ ls -1
animals.txt
myfile
myfile2
test.py
$ ls -1 | wc -l
4
```

Опция -1, ук зыв ющ я ls выводить результ ты в один столбец, здесь не обяз тельн . Чтобы узн ть, почему мы ее использов ли, см. врезку «ls меняет свое поведение при перен пр влении вывод » н с. 23.

wc — это перв я ком нд , с которой мы позн комились в этой гл ве, поэтому пок сложно использов ть к н лы. Р ди интерес н пр вим вывод wc с мой себе, чтобы продемонстриров ть, что одн и т же ком нд может появляться в конвейере более одного р з . Эт комбиниров нн я ком нд сообщ ет, что количество слов в выводе wc р вно четырем (три целых числ и имя ф йл):

```
$ wc animals.txt
  7 51 325 animals.txt
$ wc animals.txt | wc -w
4
```

Не будем ост н влив ться. Доб вим третий вызов wc в конвейер и посчит ем количество строк, слов и символов в выводе *4*:

```
$ wc animals.txt | wc -w | wc
      1       1       2
```

LS МЕНЯЕТ СВОЕ ПОВЕДЕНИЕ ПРИ ПЕРЕНАПРАВЛЕНИИ ВЫВОДА

В отличие от большинств других ком нд Linux, `ls` зн ет, выводит ли он д нные н экр н или перен пр вляет в к н л. Когд *stdout* — это экр н, `ls` упорядочив ет вывод в несколько колонок для удобств чтения:

```
$ ls /bin
bash        dir      kmod     networkctl        red      tar
bsd-csh     dmesg    less     nisdomainname     rm       tempfile
⋮
```

При перен пр влении *stdout* `ls` созд ет одну колонку. Убедимся в этом, перед в вывод `ls` ком нде, котор я воспроизводит свой ввод, т кой к к `cat`[1]:

```
$ ls /bin | cat
bash
bsd-csh
bunzip2
busybox
⋮
```

Это может привести к неожид нным результ т м, к к в следующем примере:

```
$ ls
animals.txt myfile myfile2 test.py
$ ls | wc -l
4
```

Перв я ком нд `ls` выводит все имен ф йлов в одну строку, втор я сообщ ет, что выводятся четыре строки. Если вы не зн ете об особых свойств х `ls`, это несоответствие может удивить.

`ls` позволяет переопределить свое поведение по умолч нию. Вы можете з ст вить `ls` вывести н экр н одну колонку с помощью п р метр `-1` или несколько колонок с п р метром `-C`.

Вывод ук зыв ет н одну строку (содерж щую число 4), одно слово (с мо число 4) и дв символ . Почему дв ? Потому что строк *4* з к нчив ется скрытым символом новой строки.

Дум ю, что дост точно простейших конвейеров с `wc`. По мере того к к мы изучим больше ком нд, конвейеры ст нут более пр ктичными.

[1] В з висимости от в ших н строек `ls` может использов ть т кже другие функции форм тиров ния, н пример изменение цвет при выводе н экр не. Но не при перен пр влении вывод .

Команда #2: head

Ком нд `head` выводит первые строки ф йл . Выведем первые три строки ф йл *animals.txt,* используя `head` с п р метром `-n`:

```
$ head -n3 animals.txt
python   Programming Python        2010     Lutz, Mark
snail    SSH, The Secure Shell     2005     Barrett, Daniel
alpaca   Intermediate Perl         2012     Schwartz, Randal
```

Если вы з пр шив ете больше строк, чем содержится в ф йле, `head` выведет весь ф йл (к к это дел ет `cat`). Если вы опустите п р метр `-n`, з головок по умолч нию будет состоять из 10 строк (`-n10`).

С м по себе функция `head` удобн для просмотр н ч л ф йл , когд в с не интересует ост льное содержимое. Эт ком нд р бот ет быстро и эффективно д же с очень большими ф йл ми, поскольку ей не нужно считыв ть весь ф йл. Кроме того, `head` умеет считыв ть д нные из ст нд ртного ввод и использует ст нд ртный вывод, что дел ет ее полезной в конвейер х. Подсчит ем количество слов в первых трех строк х ф йл *animals.txt*:

```
$ head -n3 animals.txt | wc -w
20
```

Обычное использов ние `head` з ключ ется в сокр щении вывод д нных от другой ком нды, когд в м не нужно видеть полную информ цию, н пример длинный список к т логов. Перечислим первые пять имен ф йлов в к т логе */bin*:

```
$ ls /bin | head -n5
bash
bsd-csh
bunzip2
busybox
bzcat
```

Команда #3: cut

Ком нд `cut` выводит одну или несколько колонок из ф йл . Н пример, выведем все н зв ния книг, которые р сположены во второй колонке ф йл *animals.txt*:

```
$ cut -f2 animals.txt
Programming Python
SSH, The Secure Shell
Intermediate Perl
MySQL High Availability
Linux in a Nutshell
```

```
Cisco IOS in a Nutshell
Writing Word Macros
```

Ком нд `cut` поддержив ет дв способ определения, что счит ть колонкой. Первый — р зделение по полям (`-f`), когд входные д нные состоят из строк (полей), к жд я из которых р зделен одним символом т буляции. Именно т кой форм т используется в ф йле *animals.txt*. Ком нд `cut` из предыдущего пример печ т ет второе поле к ждой строки бл год ря опции `-f2`.

Чтобы сокр тить вывод, перед дим его в `head`. Выведем н экр н только первые три строки:

```
$ cut -f2 animals.txt | head -n3
Programming Python
SSH, The Secure Shell
Intermediate Perl
```

Вы т кже можете вырез ть несколько полей, р зделив их номер з пятыми:

```
$ cut -f1,3 animals.txt | head -n3
python   2010
snail    2005
alpaca   2012
```

или ук з в ди п зон зн чений:

```
$ cut -f2-4 animals.txt | head -n3
Programming Python        2010     Lutz, Mark
SSH, The Secure Shell     2005     Barrett, Daniel
Intermediate Perl         2012     Schwartz, Randal
```

Т кже можно определить колонку для ком нды `cut` по положению символ в строке с использов нием п р метр `-c`. Выведем первые три символ из к ждой строки ф йл , которые можно ук з ть либо через з пятую (1, 2, 3), либо в форм те ди п зон (1–3):

```
$ cut -c1-3 animals.txt
pyt
sna
alp
rob
hor
don
ory
```

Теперь, когд вы озн комились с основными функциями, попробуем сдел ть что-нибудь более пр ктичное с помощью ком нды `cut` и с использов нием к н лов. Допустим, что ф йл *animals.txt* состоит из нескольких тысяч строк

и в м нужно извлечь только ф милии второв. Сн ч л выделим четвертое поле — имя и ф милия втор :

```
$ cut -f4 animals.txt
Lutz, Mark
Barrett, Daniel
Schwartz, Randal
⋮
```

З тем перед дим выходные д нные снов в ком нду `cut`, используя п р метр `-d` (delimiter — р зделитель), чтобы изменить символ-р зделитель н з пятую вместо т буляции. Это позволит выделить только ф милии второв:

```
$ cut -f4 animals.txt | cut -d, -f1
Lutz
Barrett
Schwartz
⋮
```

Экономьте время благодаря истории команд и редактированию

Приходилось ли в м н бир ть одни и те же ком нды з ново? Вместо этого несколько р з н жмите кл вишу со стрелкой вверх, чтобы просмотреть ком нды, которые вы з пуск ли р нее (эт функция оболочки н зыв ется *историей ком нд*). Когд дойдете до нужной ком нды, н жмите Enter, чтобы немедленно з пустить ее, или сн ч л отред ктируйте, используя кл виши со стрелк ми влево и впр во для позициониров ния курсор и кл вишу Backspace для уд ления (эт функция н зыв ется *ред ктиров ние ком ндной строки*). В гл ве 3 мы узн ем о зн чительно более мощных функциях истории ком нд и ред ктиров ния.

Команда #4: grep

`grep` — чрезвыч йно мощн я ком нд , но пок скроем большую ч сть ее возможностей и огр ничимся тем, что он печ т ет строки, соответствующие з д нному ш блону (более подробн я информ ция будет предст влен в гл ве 5). Н пример, следующ я ком нд отобр ж ет строки из ф йл *animals.txt*, содерж щие текст `Nutshell`:

```
$ grep Nutshell animals.txt
horse    Linux in a Nutshell           2009     Siever, Ellen
donkey   Cisco IOS in a Nutshell       2005     Boney, James
```

Т кже можно вывести строки, которые не соответствуют з д нному ш блону, с опцией `-v`. Обр тите вним ние, что строки, содерж щие `Nutshell`, отсутствуют:

```
$ grep -v Nutshell animals.txt
python       Programming Python         2010     Lutz, Mark
snail        SSH, The Secure Shell      2005     Barrett, Daniel
alpaca       Intermediate Perl          2012     Schwartz, Randal
robin        MySQL High Availability    2014     Bell, Charles
oryx         Writing Word Macros        1999     Roman, Steven
```

Т ким обр зом, ком нд grep полезн для поиск определенного текст в некотором списке ф йлов. Следующ я ком нд печ т ет строки, содерж щие текст Perl, в ф йл х с р сширением *.txt*:

```
$ grep Perl *.txt
animals.txt:alpaca        Intermediate Perl     2012     Schwartz, Randal
essay.txt:really love the Perl programming language, which is
essay.txt:languages such as Perl, Python, PHP, and Ruby
```

В д нном случ е ком нд grep н шл три соответствия — в одной строке ф йл *animals.txt* и в двух строк х ф йл *essay.txt*.

grep чит ет ст нд ртный ввод и з писыв ет ст нд ртный вывод, что дел ет эту ком нду иде льной для конвейеров. Допустим, мы хотим узн ть, сколько подк т логов н ходится в к т логе */usr/lib*. Нет простой ком нды Linux для получения ответ , поэтому созд дим конвейер. Н чнем с ком нды ls -l:

```
$ ls -l /usr/lib
drwxrwxr-x   12 root root   4096 Mar  1  2020 4kstogram
drwxr-xr-x   3 root root    4096 Nov 30  2020 GraphicsMagick-1.4
drwxr-xr-x   4 root root    4096 Mar 19  2020 NetworkManager
-rw-r--r--   1 root root   35568 Dec  1  2017 attica_kde.so
-rwxr-xr-x   1 root root     684 May  5  2018 cnf-update-db
⋮
```

Обр тите вним ние, что ls -l помеч ет к т логи буквой d в н ч ле строки. Используем cut, чтобы вывести первый столбец:

```
$ ls -l /usr/lib | cut -c1
d
d
d
-
-
⋮
```

З тем используем grep, чтобы ост вить только строки, содерж щие букву d:

```
$ ls -l /usr/lib | cut -c1 | grep d
d
d
d
⋮
```

Н конец, подсчит ем строки с помощью ком нды `wc` — и мы получим ответ, созд нный конвейером из четырех ком нд: */usr/lib* содержит 145 подк т логов:

```
$ ls -l /usr/lib | cut -c1 | grep d | wc -l
145
```

Команда #5: sort

Ком нд `sort` сортирует строки ф йл в порядке возр ст ния (по умолч нию):

```
$ sort animals.txt
alpaca       Intermediate Perl          2012     Schwartz, Randal
donkey       Cisco IOS in a Nutshell    2005     Boney, James
horse        Linux in a Nutshell        2009     Siever, Ellen
oryx         Writing Word Macros        1999     Roman, Steven
python       Programming Python         2010     Lutz, Mark
robin        MySQL High Availability    2014     Bell, Charles
snail        SSH, The Secure Shell      2005     Barrett, Daniel
```

или в порядке убыв ния (с п р метром `-r`):

```
$ sort -r animals.txt
snail        SSH, The Secure Shell      2005     Barrett, Daniel
robin        MySQL High Availability    2014     Bell, Charles
python       Programming Python         2010     Lutz, Mark
oryx         Writing Word Macros        1999     Roman, Steven
horse        Linux in a Nutshell        2009     Siever, Ellen
donkey       Cisco IOS in a Nutshell    2005     Boney, James
alpaca       Intermediate Perl          2012     Schwartz, Randal
```

`sort` может сортиров ть строки в лф витном порядке (по умолч нию) или в числовом порядке (с опцией `-n`). Продемонстрируем это н примере конвейеров, которые вырез ют третье поле (год публик ции) в *animals.txt*:

```
$ cut -f3 animals.txt                      несортированный
2010
2005
2012
2014
2009
2005
1999
$ cut -f3 animals.txt | sort -n            по возрастанию
1999
2005
2005
2009
2010
2012
2014
```

```
$ cut -f3 animals.txt | sort -nr            по убыванию
2014
2012
2010
2009
2005
2005
1999
```

Чтобы узнать год выхода самой новой книги в *animals.txt*, направим вывод `sort` на ввод `head` и напечатаем только первую строку:

```
$ cut -f3 animals.txt | sort -nr | head -n1
2014
```

Максимальные и минимальные значения

`sort` и `head` — мощные партнеры при работе с числовыми данными, если они расположены по одному в каждой новой строке. Вы можете вывести максимальное значение с помощью такого конвейера:

```
... | sort -nr | head -n1
```

минимальное значение с помощью следующего:

```
... | sort -n | head -n1
```

Рассмотрим другой пример с использованием файла */etc/passwd*, содержащего список пользователей, которые могут запускать процессы в системе[1]. Создадим список всех пользователей в алфавитном порядке. Первые пять строк выглядят примерно так:

```
$ head -n5 /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
smith:x:1000:1000:Aisha Smith,,,:/home/smith:/bin/bash
jones:x:1001:1001:Bilbo Jones,,,:/home/jones:/bin/bash
```

Каждая строка состоит из значений, разделенных двоеточиями, и первое из них — это имя пользователя (логин). Поэтому мы можем изолировать имена пользователей с помощью команды `cut`:

```
$ head -n5 /etc/passwd | cut -d: -f1
root
daemon
bin
smith
jones
```

[1] Некоторые системы Linux хранят информацию о пользователе в другом месте.

з тем отсортиров ть их:

```
$ head -n5 /etc/passwd | cut -d: -f1 | sort
bin
daemon
jones
root
smith
```

Созд дим сортиров нный список всех имен пользов телей, не только первых пяти из них, з менив `head` н `cat`:

```
$ cat /etc/passwd | cut -d: -f1 | sort
```

Чтобы определить, есть ли у д нного пользов теля учетн я з пись в системе, сопост вим его логин с помощью `grep`. Пустой вывод озн ч ет отсутствие учетной з писи:

```
$ cut -d: -f1 /etc/passwd | grep -w jones
jones
$ cut -d: -f1 /etc/passwd | grep -w rutabaga              пустой вывод
```

П р метр `-w` ук зыв ет ком нде `grep` сопост влять только слов целиком, не их ч сти. Если в системе имеется имя пользов теля, содерж щее *jones*, н пример `sallyjones2`, то ком нд из пример выше его не отобр зит.

Команда #6: uniq

Ком нд `uniq` обн ружив ет повторяющиеся соседние строки в ф йле. По умолч нию он уд ляет повторы. Продемонстрируем это н ф йле, содерж щем только з гл вные буквы:

```
$ cat letters
A
A
A
B
B
A
C
C
C
C
$ uniq letters
A
B
A
C
```

Обр тите вним ние, что ком нд uniq сокр тил первые три строки A до одной A, но ост вил последнюю A н месте, потому что он не был *соседней* с первыми тремя.

С помощью п р метр -c можно подсчит ть количество повторяющихся строк:

```
$ uniq -c letters
      3 A
      2 B
      1 A
      4 C
```

Призн юсь честно, когд я впервые столкнулся с ком ндой uniq, то не увидел в ней особой пользы, но он быстро ст л одной из моих любимых. Предположим, у н с есть р зделенный т буляцией ф йл с итоговыми оценк ми студентов университет в ди п зоне от A (лучш я) до F (худш я):

```
$ cat grades
C       Geraldine
B       Carmine
A       Kayla
A       Sophia
B       Haresh
C       Liam
B       Elijah
B       Emma
A       Olivia
D       Noah
F       Ava
```

Н м нужно вывести н экр н оценку, которую получил бо́льш я ч сть студентов (если будет р вное количество у нескольких оценок, в вывод поп дет перв я из них). Н чнем с извлечения столбц с оценк ми с помощью ком нды cut:

```
$ cut -f1 grades | sort
A
A
A
B
B
B
B
C
C
D
F
```

З тем используем ком нду uniq для подсчет совп д ющих строк:

```
$ cut -f1 grades | sort | uniq -c
      3 A
```

```
      4 B
      2 C
      1 D
      1 F
```

З тем отсортируем строки в порядке убыв ния, чтобы переместить н иболее ч сто встреч ющуюся оценку в верхнюю строку:

```
$ cut -f1 grades | sort | uniq -c | sort -nr
      4 B
      3 A
      2 C
      1 F
      1 D
```

и ост вим только первую строку с помощью ком нды `head`:

```
$ cut -f1 grades | sort | uniq -c | sort -nr | head -n1
      4 B
```

И н конец, поскольку н м нужн только буквенн я оценк , не количество, извлечем ее с помощью `cut`:

```
$ cut -f1 grades | sort | uniq -c | sort -nr | head -n1 | cut -c9
B
```

И вот в м ответ, бл год ря конвейеру из шести ком нд — пок что с мому длинному из использов нных н ми. Т кого род пош говое построение конвейер — не просто обуч ющее упр жнение. Именно т к н с мом деле р бот ют эксперты Linux. Этой технике посвящен гл в 8.

Обнаружение дубликатов файлов

Д в йте объединим в большом примере все то, что уже узн ли. Предположим, мы н ходимся в к т логе, который з бит ф йл ми JPEG, и хотим узн ть, не дублируются ли они:

```
$ ls
image001.jpg image005.jpg image009.jpg image013.jpg image017.jpg
image002.jpg image006.jpg image010.jpg image014.jpg image018.jpg
⋮
```

Ответить н этот вопрос можно с помощью конвейер . Н м пон добится еще одн ком нд , `md5sum`, котор я проверяет содержимое ф йл и вычисляет 32-символьную строку, н зыв емую *контрольной суммой*:

```
$ md5sum image001.jpg
146b163929b6533f02e91bdf21cb9563 image001.jpg
```

Контрольная сумма данного файла по математическим законам с очень большой вероятностью будет уникальной. Если два файла имеют одинаковую контрольную сумму, то они почти наверняка являются копиями. Здесь `md5sum` указывает, что первый и третий файлы являются копиями друг друга:

```
$ md5sum image001.jpg image002.jpg image003.jpg
146b163929b6533f02e91bdf21cb9563 image001.jpg
63da88b3ddde0843c94269638dfa6958 image002.jpg
146b163929b6533f02e91bdf21cb9563 image003.jpg
```

Повторяющиеся контрольные суммы легко обнаружить на глаз, когда файлов всего три, но что делать, если их три тысячи? На помощь приходят команды. Вычислим все контрольные суммы, с помощью `cut` изолируем первые 32 символа каждой строки и отсортируем строки, чтобы дубликаты располагались рядом друг с другом:

```
$ md5sum *.jpg | cut -c1-32 | sort
1258012d57050ef6005739d0e6f6a257
146b163929b6533f02e91bdf21cb9563
146b163929b6533f02e91bdf21cb9563
17f339ed03733f402f74cf386209aeb3
⋮
```

Далее добавим команду `uniq` для подсчета повторяющихся строк:

```
$ md5sum *.jpg | cut -c1-32 | sort | uniq -c
      1 1258012d57050ef6005739d0e6f6a257
      2 146b163929b6533f02e91bdf21cb9563
      1 17f339ed03733f402f74cf386209aeb3
      ⋮
```

Если нет дубликатов, все результаты подсчета, выведенные на экран командой `uniq`, будут равны 1. Отсортируем результаты по убыванию, и значения, которые больше 1, будут выведены в верхней части списка:

```
$ md5sum *.jpg | cut -c1-32 | sort | uniq -c | sort -nr
      3 f6464ed766daca87ba407aede21c8fcc
      2 c7978522c58425f6af3f095ef1de1cd5
      2 146b163929b6533f02e91bdf21cb9563
      1 d8ad913044a51408ec1ed8a204ea9502
      ⋮
```

Теперь давайте удалим все недублированные элементы. Их контрольным суммам предшествуют шесть пробелов, число 1 и один пробел. Мы используем `grep -v` для удаления этих строк[1]:

[1] Технически команда `sort -nr` не нужна в этом конвейере, потому что `grep` удаляет все недублированные элементы.

```
$ md5sum *.jpg | cut -c1-32 | sort | uniq -c | sort -nr | grep -v "      1 "
      3 f6464ed766daca87ba407aede21c8fcc
      2 c7978522c58425f6af3f095ef1de1cd5
      2 146b163929b6533f02e91bdf21cb9563
```

И н конец, у н с есть список повторяющихся контрольных сумм, отсортиро- в нный по количеству вхождений, созд нный изящным конвейером из шести ком нд. Если он не выводит ничего, зн чит, дублик тов ф йлов в к т логе нет.

Эт ком нд был бы еще полезней, если бы отобр ж л имен ф йлов, но для этой опер ции требуются функции, которые мы еще не обсужд ли, вы узн - ете о них в р зделе «Улучшенный способ обн ружения дублик тов ф йлов» н с. 112. А пок что идентифицируем ф йлы с з д нной контрольной суммой, выполнив поиск с помощью `grep`:

```
$ md5sum *.jpg | grep 146b163929b6533f02e91bdf21cb9563
146b163929b6533f02e91bdf21cb9563 image001.jpg
146b163929b6533f02e91bdf21cb9563 image003.jpg
```

и почистим вывод с помощью `cut`:

```
$ md5sum *.jpg | grep 146b163929b6533f02e91bdf21cb9563 | cut -c35-
image001.jpg
image003.jpg
```

Резюме

Теперь вы увидели всю мощь ст нд ртного ввод , ст нд ртного вывод и к - н лов. Они превр щ ют небольшую горстку ком нд в н бор комбинируемых инструментов, док зыв я, что целое всегд есть нечто большее, чем прост я сумм его ч стей. Люб я ком нд , котор я чит ет ст нд ртный ввод или з пи- сыв ет ст нд ртный вывод, может уч ствов ть в созд нии конвейер [1]. По мере изучения дополнительных ком нд вы сможете применять общие принципы из этой гл вы для созд ния собственных эффективных комбин ций.

[1] Некоторые ком нды не используют stdin и stdout и поэтому не могут чит ть из к н лов или з писыв ть в них. Примеры: `mv` и `rm`. Одн ко конвейеры могут использов ть эти ком нды другими способ ми, примеры р ссмотрены в гл ве 8.

ГЛАВА 2

Знакомство с командной оболочкой

Итак, вы можете запускать команды после приглашения командной строки. Но *что означает* это приглашение, откуда оно появляется и как выполняются ваши команды?

Приглашение создается *командной оболочкой* Linux. Это интерфейс пользователя, который обеспечивает взаимодействие между вами и операционной системой. Linux поддерживает несколько командных оболочек, но наиболее распространенная из них – `bash`. В этой книге по умолчанию используется именно `bash`, описание других командных оболочек вы найдете в Приложении Б.

Функции `bash` и других оболочек значительно шире, чем простое выполнение команд. Например, команда может включать в себя символ групповых операций (*) для одновременной ссылки на несколько файлов:

```
$ ls *.py
data.py main.py user_interface.py
```

Этот символ обрабатывается командной оболочкой, а не программой `ls`. Командная оболочка заменяет выражение `*.py` списком имен подходящих файлов еще *до* запуска программы `ls`. Иными словами, `ls` *не увидит* символ групповой операции. С точки зрения `ls`, вы ввели команду следующего вида:

```
$ ls data.py main.py user_interface.py
```

Командная оболочка также обрабатывает данные в каналах, с которыми вы знакомились в главе 1. Она незаметно перенаправляет стандартные потоки stdin и stdout, поэтому исполняемые программы не знают, что они взаимодействуют друг с другом.

При к ждом з пуске ком нды з одни эт пы ее выполнения отвеч ет вызыв ем я прогр мм , н пример `ls`, з другие — оболочк . Опытные пользов тели умеют созд в ть длинные сложные ком нды, поскольку *зн ют, что сдел ет ком нд* , еще до того, к к н жмут Enter, и поним ют р зделение обяз нностей между оболочкой и вызыв емыми ею прогр мм ми.

В этой гл ве мы позн комим в с с оболочкой Linux. Я буду использов ть тот же миним листский подход, что и для ком нд и конвейеров в гл ве 1. Вместо того чтобы описыв ть десятки функций оболочки, я д м в м ровно столько информ ции, сколько нужно, чтобы перейти к следующему ш гу в шего обучения:

- сопост вление ш блонов имен ф йлов;
- переменные для хр нения зн чений;
- экр ниров ние и использов ние к вычек для отключения некоторых функций оболочки;
- путь для поиск исполняемых прогр мм;
- сохр нение изменений в окружении оболочки.

Терминология командной оболочки

Слово *оболочк* употребляется в р зных зн чениях. Н пример, иногд имеется в виду *концепция* оболочки Linux в целом, к к во фр з х «ком ндн я оболочк — это мощный инструмент» или «bash — это ком ндн я оболочк ». В других случ ях речь идет о конкретном *экземпляре* оболочки н компьютере с Linux, ожид ющем в шей следующей ком нды.

В этой книге смысл понятия *оболочк* в большинстве случ ев должен быть ясен из контекст . При необходимости я буду уточнять, что имеется в виду: *экземпляр* оболочки, *р бот ющ я* или в ш *текущ я* ком ндн я оболочк .

Некоторые экземпляры оболочки предост вляют пользов телю пригл шение ком ндной строки, чтобы он мог вз имодейств ть с ними. Для обозн чения этих экземпляров я буду использов ть термин *интер ктивн я оболочк* . Другие экземпляры оболочки не интер ктивны — они выполняют последов тельность ком нд и з верш ют р боту.

Сопоставление шаблонов имен файлов

В гл ве 1 вы р бот ли с ком нд ми `cut`, `sort` и `grep`, которые могут приним ть одно или несколько имен ф йлов в к честве ргументов. Н пример, вы можете иск ть слово Linux в сотне ф йлов с имен ми от *chapter1* до *chapter100*:

```
$ grep Linux chapter1 chapter2 chapter3 chapter4 chapter5...
```

Перечислять множество имен ф йлов — довольно утомительное з нятие. Поэтому оболочк позволяет использов ть специ льные символы для сокр щений, которые применяются к ф йл м и к т лог м. Ч сто их н зыв ют символ ми групповых опер ций, но более общ я концепция н зыв ется *сопост влением с ш блоном (pattern matching)* или *подст новкой (globbing)*. Сопост вление с ш блоном — это один из двух н иболее популярных приемов для повышения скорости р боты в Linux (другой прием — н ж ть кл вишу со стрелкой вверх, чтобы вызв ть предыдущую ком нду оболочки, — мы р смотрим его в гл ве 3).

В Linux символ звездочки `*` соответствует любой последов тельности (з исключением н ч льной точки в имен х ф йлов или к т логов)[1] из любого числ символов в путях к ф йл м или к т лог м:

```
$ grep Linux chapter*
```

Ост ется з к дром, к к ком ндн я оболочк (не прогр мм `grep`!) преобр зует ш блон `chapter*` в список подходящих имен ф йлов. И только после этого оболочк з пуск ет `grep`.

Еще один специ льный символ — зн к вопрос `?`, который соответствует любому единичному символу, з исключением н ч льной точки в имен х ф йлов или к т логов. Н пример, вы можете выполнить поиск слов *Linux* в гл в х с 1 по 9, используя вместо цифр зн к вопрос , чтобы ком ндн я оболочк смогл н йти совп дения:

```
$ grep Linux chapter?
```

Для поиск в гл в х с 10 по 99 придется использов ть дв зн к вопрос :

```
$ grep Linux chapter??
```

---

[1] По этой причине ком нд `ls *` не отобр зит список т к н зыв емых скрытых ф йлов и к т логов (dotfiles), которые н чин ются с точки.

Менее известно использов ние кв др тных скобок [] для з прос у ком ндной оболочки соответствия одному из символов н бор . Н пример, вы можете иск ть только в первых пяти гл в х:

```
$ grep Linux chapter[12345]
```

З д ть ди п зон можно с помощью тире:

```
$ grep Linux chapter[1-5]
```

Вы т кже можете иск ть гл вы с четными номер ми, комбинируя звездочку и кв др тные скобки, чтобы оболочк иск л соответствия имен м ф йлов, ок нчив ющимся н четную цифру:

```
$ grep Linux chapter*[02468]
```

Любые символы, не только цифры, могут быть помещены в кв др тные скобки для сопост вления. Н пример, следующ я ком нд з ст вит оболочку иск ть имен ф йлов, н чин ющиеся с з гл вной буквы, содерж щие символ подчеркив ния и з к нчив ющиеся символом @:

```
$ ls [A-Z]*_*@
```

Терминология: вычисление выражений и расширение шаблонов

Строки, которые вы вводите в ком ндной строке, н пример `chapter*` или `Efficient linux`, н зыв ются *выр жениями*. Ком нд , н пример `ls -l chapter*`, т кже является выр жением.

Когд оболочк интерпретирует и обр б тыв ет специ льные символы в выр жении, т кие к к звездочк и вертик льн я черт , мы говорим, что оболочк *вычисляет* выр жение.

Сопост вление с ш блоном — это один из видов вычислений. Когд ком ндн я оболочк вычисляет выр жение, которое содержит символы групповых опер ций, н пример `chapter*`, и з меняет его имен ми ф йлов, соответствующими ш блону, мы говорим, что оболочк *р сширяет* ш блон.

Ш блоны применимы пр ктически всегд , когд вы ук зыв ете пути к ф йл м или к т лог м в ком ндной строке. Н пример, вы можете перечислить все ф йлы в к т логе */etc* с имен ми, з к нчив ющимися н *.conf*, следующим обр зом:

```
$ ls -1 /etc/*.conf
/etc/adduser.conf
/etc/appstream.conf
```

```
⋮
/etc/wodim.conf
```

Будьте осторожны, используя ш блон с ком ндой, котор я приним ет только один ргумент — имя ф йл или к т лог . Н пример, с ком ндой `cd` вы можете получить совсем не то, что ожид ли:

```
$ ls
Pictures Poems Politics
$ cd P*                 Будет найдено три подходящих каталога
bash: cd: too many arguments
```

Если ш блон не соответствует ни одному ф йлу, оболочк перед ет его в к честве ргумент ком нды. В ком нде ниже ш блон `*.doc` не соответствует ни одному ф йлу в текущем к т логе, поэтому ком нд `ls` ищет ф йл с именем `*.doc` и сообщ ет о неуд че:

```
$ ls *.doc
/bin/ls: cannot access '*.doc': No such file or directory
```

При р боте с ш блон ми в жно помнить дв момент . Во-первых, к к я уже подчеркив л, сопост вление с обр зцом выполняет ком ндн я оболочк , не вызыв ем я прогр мм . Удивительно, что многие пользов тели Linux не зн ют об этом и недоумев ют, почему одни ком нды выполняются успешно, другие — нет.

Второй в жный момент з ключ ется в том, что сопост вление с обр зцом р бот ет только с имен ми ф йлов и к т логов. Оно неприменимо для имен пользов телей, хостов и других типов ргументов. Вы т кже не можете ввести, н пример, `s?rt` в н ч ле ком ндной строки и ожид ть, что оболочк з пустит прогр мму сортировки (некоторые ком нды Linux, т кие к к `grep`, `sed` и `awk`, поддержив ют особые виды сопост вления с ш блоном, которые мы р ссмотрим в гл ве 5).

Сопоставление имен файлов с шаблоном и ваши программы

Все прогр ммы, которые приним ют имен ф йлов в к честве ргументов, втом тически поддержив ют сопост вление с ш блоном, поскольку ком ндн я оболочк обр б тыв ет ш блоны до з пуск прогр ммы. Это верно и для н пис нных в ми прогр мм и скриптов. Н пример, вы н пис ли прогр мму `english2swedish`, котор я переводит ф йлы с нглийского язык н шведский и приним ет несколько имен ф йлов в ком ндной строке. Вы можете без проблем з пустить ее с ш блоном:

```
$ english2swedish *.txt
```

Вычисление переменных

Ком ндн я оболочк может определять переменные и сохр нять в них зн чения. Переменн я оболочки очень похож н переменную в лгебре — у нее есть имя и зн чение. Примером может служить переменн я оболочки `HOME`. Ее зн чением является путь к в шему дом шнему к т логу Linux, н пример */home/smith*. Другой пример — переменн я `USER`, зн чением которой является в ше имя пользов теля Linux. В этой книге `smith` будет использов ться к к зн чение переменной `USER`.

Для вывод н экр н зн чений переменных `HOME` и `USER` з пустите ком нду `printenv`:

```
$ printenv HOME
/home/smith
$ printenv USER
smith
```

Когд ком ндн я оболочк вычисляет переменную, он з меняет имя переменной н ее зн чение. Чтобы вычислить переменную, просто поместите зн к долл р перед именем. Н пример, `$HOME` возвр щ ет строку */home/smith*.

С мый простой способ увидеть, к к оболочк обр б тыв ет ком ндную строку, — з пустить ком нду `echo`, котор я выводит свои ргументы (после того к к оболочк з вершит их вычисление):

```
$ echo My name is $USER and my files are in $HOME        Вычисление переменных
My name is smith and my files are in /home/smith
$ echo ch*ter9                                           Вычисление шаблона
chapter9
```

Откуда берутся значения переменных

Т кие переменные, к к `USER` и `HOME`, предопределены оболочкой. Их зн чения уст н влив ются втом тически при входе в систему (подробности об этом мы узн ем позже). Имен предопределенных переменных тр диционно н бир ются прописными букв ми.

Вы можете определить или изменить переменную в любое время: присвоить ей зн чение, используя следующий синт ксис:

```
name=value
```

Н пример, если вы ч сто р бот ете с к т логом */home/smith/Projects*, вы можете присвоить его имя переменной:

```
$ work=$HOME/Projects
```

и использов ть его к к удобное сокр щение при р боте с cd:

```
$ cd $work
$ pwd
/home/smith/Projects
```

Вы можете перед в ть $work любой ком нде, ожид ющей имя к т лог , в к честве ргумент :

```
$ cp myfile $work
$ ls $work
Myfile
```

При определении переменной не допуск ется использов ние пробелов около зн к р венств . Если вы об этом з будете, оболочк примет (ошибочно), что первое слово в ком ндной строке — это з пуск ем я прогр мм , зн к р венств и зн чение — ее ргументы, и вы увидите сообщение об ошибке:

```
$ work = $HOME/Projects          Командная оболочка решила, что «work» – это команда
work: command not found
```

Пр вил р боты с определяемыми пользов телями переменными, к к, н пример, work, — т кие же, к к и с системными переменными вроде HOME. Единственное отличие состоит в том, что некоторые прогр ммы Linux меняют свое поведение н основе зн чений HOME, USER и других *системных* переменных. Н пример, прогр мм Linux с гр фическим интерфейсом может получить в ше имя пользов теля из оболочки и отобр зить его. Т кие прогр ммы не обр щ ют вним ния н созд нную пользов телем переменную, т кую, н пример, к к work.

Переменные и заблуждения

Когд вы выводите н экр н зн чение переменной с помощью ком нды echo:

```
$ echo $HOME
/home/smith
```

вы можете подум ть, что ком нд echo проверяет переменную HOME и выводит ее зн чение. Н с мом деле *это не т к*. Ком нд echo ничего не зн ет о переменных. Он просто выводит н экр н любые ргументы, которые вы ей перед ете. Зн чение $HOME вычисляет ком ндн я оболочк перед з пуском ком нды echo. С точки зрения echo, вы н бр ли:

```
$ echo /home/smith
```

Этот принцип в жно понять перед тем, к к мы углубимся в более сложные ком нды. Оболочк вычисляет переменные, ш блоны и другие конструкции перед выполнением ком нды.

Шаблоны vs переменные

Д в йте проверим в ше поним ние ш блонов и вычисления переменных. Предположим, вы н ходитесь в к т логе с двумя подк т лог ми, *mammals* и *reptiles*[1]. И, к к ни стр нно, в подк т логе *mammals* н ходятся ф йлы *lizard.txt* и *snake.txt*[2]:

```
$ ls
mammals reptiles
$ ls mammals
lizard.txt snake.txt
```

Поскольку ящерицы и змеи не являются млекопит ющими, эти дв ф йл следует переместить в подк т лог *reptiles*. Вот дв способ сдел ть это, но один р бот ет, другой нет:

```
mv mammals/*.txt reptiles      Метод 1


FILES="lizard.txt snake.txt"
mv mammals/$FILES reptiles     Метод 2
```

Метод 1 р бот ет, потому что ш блон соответствует всему пути к ф йлу — имя к т лог *mammals* является ч стью обоих совп дений для `mammals/*.txt`:

```
$ echo mammals/*.txt
mammals/lizard.txt mammals/snake.txt
```

Т ким обр зом, метод 1 р бот ет т к, к к если бы вы н бр ли следующую корректную ком нду:

```
$ mv mammals/lizard.txt mammals/snake.txt reptiles
```

В методе 2 используются переменные, имеющие только свои букв льные зн чения, и нет специ льного инструмент для вычисления путей к ф йл м:

```
$ echo mammals/$FILES
mammals/lizard.txt snake.txt
```

Следов тельно, метод 2 р бот ет т к, к к если бы вы н бр ли следующую не совсем корректную ком нду:

```
$ mv mammals/lizard.txt snake.txt reptiles
```

Эт ком нд ищет ф йл *snake.txt* в текущем к т логе, не в к т логе *mammals*, и выд ет ошибку:

---

[1] Mammals – млекопит ющие, reptiles — рептилии. — *Примеч. перев.*

[2] Lizard — ящериц , snake —змея. — *Примеч. перев.*

```
$ mv mammals/$FILES reptiles
/bin/mv: cannot stat 'snake.txt': No such file or directory
```

Чтобы сдел ть метод 2 р бочим, используйте цикл `for`, который доб вляет имя к т лог *mammals* к к ждому имени ф йл :

```
FILES="lizard.txt snake.txt"
for f in $FILES; do
mv mammals/$f reptiles
done
```

Сокращение команд с помощью псевдонимов

Переменн я — это имя, которому соответствует определенное зн чение. В ком ндной оболочке т кже есть имен , которым соответствуют ком нды. Они н зыв ются *псевдоним ми* (*aliases*). Определим псевдоним, пост вив после него зн к р венств и ком нду:

```
$ alias g=grep                    Пример команды без аргументов
$ alias ll="ls -l"                Пример команды с аргументами, кавычки необходимы
```

З пустите псевдоним, введя его имя в к честве ком нды. Когд псевдонимы короче ком нд, которые они вызыв ют, вы экономите время н н боре текст :

```
$ ll                                        Запускается «ls -l»
-rw-r--r-- 1 smith smith 325 Jul 3 17:44 animals.txt
$ g Nutshell animals.txt                    Запускается «grep Nutshell animals.txt»
horse    Linux in a Nutshell       2009   Siever, Ellen
donkey   Cisco IOS in a Nutshell   2005   Boney, James
```

Всегд определяйте псевдоним в отдельной строке, не к к ч сть комбиниров нной ком нды (технические подробности смотрите в `man bash`).

Вы можете определить псевдоним с тем же именем, что и у существующей ком нды, ф ктически з менив эту ком нду в в шей оболочке. Эт пр ктику можно н зв ть *з тенением* (*shadowing*) ком нды. Предположим, в м нр вится ком нд `less` для чтения ф йлов, но вы хотите, чтобы он очищ л экр н перед отобр жением к ждой стр ницы. Эт функция ктивируется опцией `-c`, поэтому определите псевдоним с именем `less`, который з пуск ет `less -c`[3]:

```
$ alias less="less -c"
```

[3] `bash` предотвр щ ет бесконечную рекурсию, не р сширяя второе зн чение `less` к к псевдоним.

Псевдонимы имеют приоритет н д ком нд ми с тем же именем, следов тельно, теперь вы з тенили ком нду `less` в текущей оболочке. К вопросу о приоритете мы вернемся в р зделе «Путь для поиск и псевдонимы».

Чтобы вывести все псевдонимы и их зн чения, которые определены н д нный момент в в шей ком ндной оболочке, з пустите `alias` без ргументов:

```
$ alias
alias g='grep'
alias ll='ls -l'
```

Чтобы вывести зн чение конкретного псевдоним , з пустите `alias` и перед йте его имя в к честве ргумент :

```
$ alias g
alias g='grep'
```

Чтобы уд лить псевдоним из оболочки, выполните `unalias`:

```
$ unalias g
```

Перенаправление ввода и вывода

Оболочк упр вляет вводом и выводом з пуск емых ком нд. Вы уже видели пример к н лов, которые н пр вляют ст нд ртный вывод одной ком нды в ст нд ртный ввод другой. Использов ние вертик льной черты | — это особенность оболочки.

Еще одн функция оболочки — перен пр вление ст нд ртного вывод в ф йл. Н пример, если вы используете `grep` для поиск подходящих строк в ф йле *animals.txt* из пример 1.1, то по умолч нию ком нд н пр вит их в ст нд ртный вывод:

```
$ grep Perl animals.txt
alpaca Intermediate Perl    2012  Schwartz, Randal
```

Вместо этого вы можете н пр вить вывод ком нды в ф йл, используя функцию оболочки, н зыв емую *перен пр влением вывод* (*output redirection*). Просто доб вьте символ > с именем ф йл , в который будет перен пр влен вывод:

```
$ grep Perl animals.txt > outfile      на экран ничего не выводится
$ cat outfile
alpaca Intermediate Perl    2012  Schwartz, Randal
```

Вы только что перен пр вили ст нд ртный вывод ком нды в ф йл *outfile* вместо вывод его н экр н. Если ф йл *outfile* не существует, он будет созд н. Если же он

существует, перен пр вление перез пишет его содержимое. Если вы хотите доб влять д нные в ф йл, не перез писыв ть его к ждый р з, используйте символ >>:

```
$ grep Perl animals.txt > outfile           Создание или перезапись файла outfile
$ echo There was just one match >> outfile  Добавление записи в конец файла outfile
$ cat outfile
alpaca Intermediate Perl      2012     Schwartz, Randal
There was just one match
```

Помимо перен пр вления вывод , возможно и *перен пр вление ввод* (*input redirection*), обеспечив ющее ст нд ртный ввод из ф йл , не с кл ви туры. Используйте символ < и имя ф йл , чтобы перен пр вить ст нд ртный ввод.

Многие ком нды Linux, которые приним ют в к честве ргументов имен ф йлов и з тем считыв ют д нные из них, т кже могут получ ть д нные из ст нд ртного ввод . Примером является ком нд wc для подсчет строк, слов и символов в ф йле:

```
$ wc animals.txt                            Чтение из файла
  7 51 325 animals.txt
$ wc < animals.txt                          Чтение из перенаправленного ввода
  7 51 325
```

Очень в жно поним ть, чем отлич ется поведение этих двух ком нд wc:

- В первом случ е ком нд wc получ ет в к честве ргумент имя ф йл *animals.txt*, поэтому он н ходит, открыв ет и чит ет содержимое ф йл н диске.
- Во втором случ е ком нд wc вызыв ется без ргументов, поэтому они считыв ются со ст нд ртного ввод . Обычно ст нд ртный ввод выполняется с кл ви туры, но ком ндн я оболочк перен пр вляет его. При этом wc понятия не имеет о существов нии ф йл *animals.txt*.

Ком ндн я оболочк может перен пр влять ввод и вывод в р мк х одной ком нды:

```
$ wc < animals.txt > count
$ cat count
  7 51 325
```

и д же может использов ть к н лы одновременно. Ниже grep чит ет из перен пр вленного ст нд ртного ввод и перед ет результ ты в ком нду wc, котор я перен пр вляет ст нд ртный вывод в ф йл *count*, созд в я его:

```
$ grep Perl < animals.txt | wc > count
$ cat count
      1       6      47
```

Т кие комбиниров нные ком нды будут дет льно р ссмотрены в гл ве 8, но примеры перен пр вления будут встреч ться во всей книге.

СТАНДАРТНЫЙ ПОТОК ОШИБОК (STDERR) И ЕГО ПЕРЕНАПРАВЛЕНИЕ

Некоторые выходные д нные, т кие к к сообщения об ошибк х, не могут быть перен пр влены с помощью >. Н пример, попросите cp скопиров ть несуществующий ф йл — и он выд ст следующее сообщение об ошибке:

```
$ cp nonexistent.txt file.txt
cp: cannot stat 'nonexistent.txt': No such file or directory
```

Если вы перен пр вите ст нд ртный вывод т кой ком нды cp в ф йл, н пример *errors*, то сообщение все р вно будет выведено н экр н:

```
$ cp nonexistent.txt file.txt > errors
cp: cannot stat 'nonexistent.txt': No such file or directory
```

ф йл *errors* ост нется пустым:

```
$ cat errors                                    Ничего не выводит
```

Почему т к происходит? Ком нды Linux могут созд в ть более одного поток вывод . В дополнение к stdout есть т кже stderr (ст нд ртный поток ошибок) — второй поток вывод , который тр диционно з резервиров н для сообщений об ошибк х. Потоки stderr и stdout выглядят один ково н дисплее, но по своей сути они р зличны. Вы можете перен пр вить stderr с помощью символов 2>, з которым следует имя ф йл :

```
$ cp nonexistent.txt file.txt 2> errors
$ cat errors
cp: cannot stat 'nonexistent.txt': No such file or directory
```

или же сообщение об ошибке можно допис ть в конец ф йл с помощью 2>>:

```
$ cp nonexistent.txt file.txt 2> errors
$ cp another.txt file.txt 2>> errors
$ cat errors
cp: cannot stat 'nonexistent.txt': No such file or directory
cp: cannot stat 'another.txt': No such file or directory
```

Для перен пр вления обоих потоков stdout и stderr в один ф йл используйте символы &>:

```
$ echo This file exists > goodfile.txt        Создание файла
$ cat goodfile.txt nonexistent.txt &> all.output
$ cat all.output
This file exists
cat: nonexistent.txt: No such file or directory
```

Отключение вычисления с помощью кавычек и экранирования

Обычно оболочк использует пробелы в к честве р зделителей между слов ми. Следующ я ком нд состоит из четырех слов — имени прогр ммы, з которым следуют три ргумент :

```
$ ls file1 file2 file3
```

Одн ко иногд нужно, чтобы оболочк воспринim л пробелы к к символы, не к к р зделители. Типичным примером являются пробелы в имени ф йл , н пример *Efficient Linux Tips.txt*:

```
$ ls -l
-rw-r--r-- 1 smith smith 36 Aug 9 22:12 Efficient Linux Tips.txt
```

Если вы обр титесь к ф йлу с т ким именем в ком ндной строке, в ш ком нд , скорее всего, з вершится ошибкой, поскольку оболочк интерпретирует символы пробел к к р зделители:

```
$ cat Efficient Linux Tips.txt
cat: Efficient: No such file or directory
cat: Linux: No such file or directory
cat: Tips.txt: No such file or directory
```

Чтобы з ст вить оболочку р ссм трив ть пробелы к к ч сть имени ф йл , у в с есть три в ри нт — один рные к вычки, двойные к вычки и обр тн я кос я черт :

```
$ cat 'Efficient Linux Tips.txt'
$ cat "Efficient Linux Tips.txt"
$ cat Efficient\ Linux\ Tips.txt
```

Один рные к вычки сообщ ют ком ндной оболочке, что к ждый символ в строке следует обр б тыв ть букв льно, д же если он имеет особое зн чение для оболочки, к к, н пример, пробелы и зн ки долл р :

```
$ echo '$HOME'
$HOME
```

Двойные к вычки ук зыв ют оболочке воспринim ть все символы букв льно, з исключением зн ков долл р и некоторых других, о которых вы узн ете позже:

```
$ echo "Notice that $HOME is evaluated"          Двойные кавычки
Notice that /home/smith is evaluated
$ echo 'Notice that $HOME is not'                 Одинарные кавычки
Notice that $HOME is not
```

Обр тн я кос я черт , т кже н зыв ем я *экр нирующим символом*, ук зыв ет оболочке восприним ть букв льно символ, н ходящийся после нее. В следующей ком нде экр нируется зн к долл р :

```
$ echo \$HOME
$HOME
```

Обр тн я кос я черт действует к к экр нирующий символ в двойных к вычк х:

```
$ echo "The value of \$HOME is $HOME"
The value of $HOME is /home/smith
```

но не в один рных:

```
$ echo 'The value of \$HOME is $HOME'
The value of \$HOME is $HOME
```

Можно использов ть обр тную косую черту, чтобы экр ниров ть символ двойных к вычек внутри двойных к вычек:

```
$ echo "This message is \"sort of\" interesting"
This message is "sort of" interesting
```

Обр тн я кос я черт в конце строки отключ ет функцию невидимого символ новой строки, позволяя ком нд м оболочки з ним ть несколько строк:

```
$ echo "This is a very long message that needs to extend \
onto multiple lines"
This is a very long message that needs to extend onto multiple lines
```

Обр тн я кос я черт в конце строки — это прекр сный способ сдел ть конвейер более чит емым, к к, ск жем, вот этот пример из подр здел «Ком нд #6: uniq» со с. 30:

```
$ cut -f1 grades \
  | sort \
  | uniq -c \
  | sort -nr \
  | head -n1 \
  | cut -c9
```

При т ком использов нии обр тную косую черту иногд н зыв ют символом продолжения строки.

Н ч льн я обр тн я кос я черт перед псевдонимом экр нирует его, з ст вляя оболочку иск ть ком нду с тем же именем. З тенение при этом игнорируется:

```
$ alias less="less -c"          Определение псевдонима
$ less myfile                   Запуск псевдонима, который определен как less -c
$ \less myfile                  Запуск стандартной команды less, а не псевдонима
```

Расположение исполняемых программ

Когд оболочк впервые встреч ет простую ком нду, н пример `ls *.py`, то р збив ет строку н дв слов : `ls` и `*.py`. В этом случ е первое слово — это имя прогр ммы н диске, и оболочк должн ее н йти, чтобы з пустить.

Ок зыв ется, прогр мм `ls` предст вляет собой исполняемый ф йл в к т логе */bin*. Вы можете проверить ее местоположение с помощью ком нды:

```
$ ls -l /bin/ls
-rwxr-xr-x 1 root root 133792 Jan 18 2018 /bin/ls
```

или изменить текущий к т лог с помощью ком нды `cd /bin` и з пустить з - г дочную ком нду:

```
$ ls ls
ls
```

котор я использует прогр мму `ls` для отобр жения исполняемого ф йл *ls*.

К к оболочк н ходит `ls` в к т логе */bin*? З к дром ост ется то, к к оболочк сверяется с предв рительно сост вленным списком к т логов, хр нящимся в ее п мяти, который н зыв ется *путем для поиск исполняемых прогр мм*. Этот список хр нится к к зн чение переменной оболочки `PATH`:

```
$ echo $PATH
/home/smith/bin:/usr/local/bin:/usr/bin:/bin:/usr/games:/usr/lib/java/bin
```

К т логи в пути для поиск р зделяются двоеточиями (:). Для большей н глядности преобр зуйте двоеточия в переводы строки, перед в вывод ком нде `tr`, котор я з меняет символы (подробнее см. гл ву 5):

```
$ echo $PATH | tr : "\n"
/home/smith/bin
/usr/local/bin
/usr/bin
/bin
/usr/games
/usr/lib/java/bin
```

Оболочк просм трив ет к т логи в списке, пыт ясь н йти прогр мму `ls`. «Существует ли */home/smith/bin/ls*? Нет. Существует ли */usr/local/bin/ls*? Нет. К к н счет */usr/bin/ls*? Снов нет! Может, *//bin/ls*? Д , вот оно! З пуск ю */bin/ls*». Этот поиск происходит слишком быстро, чтобы можно было его з метить[1].

[1] Некоторые оболочки з помин ют (кэшируют) пути к прогр мм м по мере их обн ружения, сокр щ я количество поисковых з просов в будущем.

Чтобы найти программу в списке каталогов для поиска, используйте команду `which`:

```
$ which cp
/bin/cp
$ which which
/usr/bin/which
```

или встроенную в оболочку более мощную и информативную команду `type`, которая также находит псевдонимы, функции и встроенные в оболочку команды[1].

```
$ type cp
cp is hashed (/bin/cp)
$ type ll
ll is aliased to '/bin/ls -l'
$ type type
type is a shell builtin
```

Если есть программы с одинаковыми названиями, расположенные в разных каталогах, например */usr/bin/less* и */bin/less*, оболочка запустит тот файл, который найдет первым. Используя эту особенность, вы можете переопределить команду, поместив соответствующий файл в каталог, который будет просматриваться раньше, например в вашем личный каталог *$HOME/bin*.

Путь для поиска и псевдонимы

Когда оболочка ищет команду по имени, она прежде всего проверяет, является ли это имя псевдонимом. Именно поэтому псевдоним затеняет команду (имеет более высокий приоритет) с таким же именем.

Путь для поиска — отличный пример того, как можно найти простое объяснение загадочному явлению. Оболочка не берет команды из воздуха и не находит их по волшебству. Она методично просматривает каталоги в списке, пока не найдет запрошенный исполняемый файл.

Окружение и файлы инициализации, краткая версия

Запущенная оболочка хранит в переменных множество важной информации: путь для поиска, текущий каталог, предпочитаемый вами текстовый редактор,

[1] Обратите внимание, что команда `type which` записывает в стандартный вывод данные, команда `which type` этого не делает.

н стр ив емое приветствие ком ндной строки и многое другое. Переменные з - пущенной ком ндной оболочки в совокупности н зыв ются *окружением*. Когд пользов тель выходит из ком ндной оболочки, ее окружение уничтож ется.

Было бы чрезвыч йно утомительно при к ждом з пуске оболочки определять ее окружение вручную. Поэтому оно определяется в сцен риях, н зыв емых *ф йл ми з пуск* и *ф йл ми иници лиз ции*, которые втом тически выполняются при з пуске ком ндной оболочки. В результ те определенн я информ ция к жется «глоб льной» или «известной» всем з пущенным экземпляр м оболочки.

Мы р ссмотрим тонкости этой темы в р зделе «Н стройк окружения» н с. 129. А здесь р сск жем только об одном ф йле иници лиз ции, чтобы вы могли понять следующие несколько гл в. Он н ходится в в шем дом шнем к т логе и н зыв ется *.bashrc*. Поскольку имя н чин ется с точки, `ls` по умолч нию не отобр ж ет его:

```
$ ls $HOME
apple  banana  carrot
$ ls -a $HOME
.bashrc  apple  banana  carrot
```

Если *$HOME/.bashrc* не существует, созд йте его с помощью текстового ред к- тор . Ком нды, которые вы поместите в этот ф йл, будут выполняться втом - тически при з пуске экземпляр оболочки[2], поэтому он отлично подойдет для определения переменных окружения и псевдонимов. Вот пример ф йл *.bashrc* (строки, н чин ющиеся с #, являются коммент риями):

```
# Задать путь для поиска
PATH=$HOME/bin:/usr/local/bin:/usr/bin:/bin
# Задать приглашение командной строки
PS1='$ '
# Задать предпочитаемый текстовый редактор
EDITOR=emacs
# Начать в моем рабочем каталоге
cd $HOME/Work/Projects
# Определить псевдоним
alias g=grep
# Сердечно поприветствовать
echo "Welcome to Linux, friend!"
```

Изменения, которые вы вносите в *$HOME/.bashrc*, не влияют н з пущенные экземпляры оболочки. Но вы можете з ст вить з пущенный экземпляр з ново выполнить *$HOME/.bashrc* с помощью одной из следующих ком нд:

```
$ source $HOME/.bashrc          Используется встроенная команда source
$ . $HOME/.bashrc               Используется точка
```

[2] Это очень упрощенное утверждение; более подробн я информ ция предст влен в т бл. 6.1.

Этот процесс известен к к *считыв ние* ф йл иници лиз ции. Говоря «использов ть исходный ф йл .bashrc», имеют в виду выполнение одной из предыдущих ком нд.

В ре льной жизни не помещ йте всю конфигур цию в шей оболочки в *$HOME/.bashrc*. Прочит в подробности в р зделе «Н стройк окружения» н с. 129, изучите ф йл *$HOME/.bashrc* и при необходимости переместите ком нды в соответствующие ф йлы.

Резюме

Мы р ссмотрели только немногие функции `bash` и их основные способы применения. Вы узн ете гор здо больше в следующих гл в х, особенно в гл ве 6. Сейч с в ш основн я з д ч — усвоить следующие принципы р боты оболочки:

- Оболочк выполняет в жные функции.
- Оболочк интерпретирует ком ндную строку перед з пуском любых ком нд.
- Ком нды могут перен пр влять ст нд ртные потоки ввод , вывод и ошибок.
- К вычки и экр ниров ние предотвр щ ют интерпрет цию ком ндной оболочкой специ льных символов.
- Оболочк н ходит прогр ммы, используя путь для поиск ф йлов.
- Вы можете изменить н стройки ком ндной оболочки, доб вив ком нды в ф йл *$HOME/.bashrc*.

Чем лучше вы будете поним ть р зделение обяз нностей между оболочкой и прогр мм ми, которые он вызыв ет, тем больше смысл будет иметь для в с ком ндн я строк и тем лучше вы сможете предск з ть, что произойдет, прежде чем вы н жмете Enter для з пуск ком нды.

ГЛАВА 3

Повторный запуск команд

Предположим, вы только что выполнили длинную ком нду со сложным конвейером, к к, н пример, следующ я ком нд из р здел «Обн ружение дублик тов ф йлов» (см. с. 32)

```
$ md5sum *.jpg | cut -c1-32 | sort | uniq -c | sort -nr
```

и вы хотите з пустить ее еще р з. Только не печ т йте ее з ново! Вместо этого попросите оболочку обр титься к истории. З к дром ост ется то, к к оболочк ведет з пись вызыв емых ком нд, но вы можете повторно з пуск ть их несколькими н ж тиями кл виш. Эт функция оболочки н зыв ется *историей ком нд*. Опытные пользов тели Linux используют ее, чтобы ускорить свою р боту и не тр тить время впустую.

Точно т к же предположим, что вы допустили ошибку при н боре предыдущей ком нды, н пример н пис в `jg` вместо `jpg`:

```
$ md5sum *.jg | cut -c1-32 | sort | uniq -c | sort -nr
```

Чтобы испр вить ошибку, не н жим йте кл вишу Backspace десятки р з и не вводите все з ново. Вместо этого просто измените ком нду. Оболочк поддержив ет *ред ктиров ние ком ндной строки* для испр вления опеч ток и выполнения всевозможных модифик ций, т к же к к это дел ет текстовый ред ктор.

Эт гл в пок жет в м, к к сэкономить время и силы при н боре текст , используя историю ком нд и ред ктиров ние ком ндной строки. К к обычно, я не буду пыт ться объять необъятное, сосредоточусь н н иболее пр ктичных и полезных ч стях этих функций ком ндной оболочки (если вы используете оболочку, отличную от `bash`, дополнительные примеч ния см. в Приложении Б).

Научитесь печатать вслепую

Зн ния из этой книги послуж т в м лучше, если вы сможете быстро печ т ть. Если вы печ т ете 40 слов в минуту, в ш не менее зн ющий друг н бир ет 120, зн чит, он способен р бот ть в три р з быстрее, чем вы. Поищите в интернете «тест скорости н бор текст », чтобы измерить свою скорость, з тем поищите «кл ви турный трен жер» и выр бот йте этот полезный н вык н всю жизнь. Попробуйте достичь 100 слов в минуту. Результ т стоит потр ченных усилий.

Просмотр истории команд

История ком нд — это список ком нд, которые вы выполняли в интер ктивной оболочке. Чтобы просмотреть историю, з пустите ком нду `history`, встроенную в оболочку. Ком нды отобр ж ются в хронологическом порядке с идентифик ционными номер ми для удобств . Вывод выглядит примерно т к:

```
$ history
1000  cd $HOME/Music
1001  ls
1002  mv jazz.mp3 jazzy-song.mp3
1003  play jazzy-song.mp3
⋮                                    Пропустим 479 строк
1481  cd
1482  firefox https://google.com
1483  history                        Включая только что запущенную команду
```

Вывод `history` может состоять из сотен (и более) строк. Огр ничьте его с мыми последними ком нд ми, доб вив целочисленный ргумент, который ук зыв ет количество выводимых строк:

```
$ history 3                          Выведем три последние команды
1482  firefox https://google.com
1483  history
1484  history 3
```

Поскольку `history` з писыв ет в ст нд ртный вывод, вы можете обр б тыв ть ее вывод с помощью конвейеров. Н пример, просмотрите свою историю по одному экр ну з р з:

```
$ history | less                 От самой последней команды к первой запущенной
$ history | sort -nr | less      От самой первой команды к последней запущенной
```

или выведите только ком нды, содерж вшие cd:

```
$ history | grep -w cd
1000  cd $HOME/Music
1092  cd ..
1123  cd Finances
1375  cd Checking
1481  cd
1485  history | grep -w cd
```

Чтобы очистить историю для текущего экземпляр оболочки, используйте п р метр -c:

```
$ history -c
```

Повторный вызов команд из истории

Р ссмотрим три экономящих время способ вызов ком нд из истории оболочки.

Перемещение курсор

Чрезвыч йно прост в освоении, но ч сто неудобен н пр ктике.

Р сширение истории

Сложнее в освоении (честно говоря, причины этого з г дочны), но может быть очень быстрым.

Инкремент льный поиск

И просто, и быстро.

К ждый метод удобен для определенных ситу ций, поэтому рекомендуется изучить все три. Чем больше техник вы зн ете, тем легче подберете нужную в любой ситу ции.

Перемещение курсора по истории команд

Чтобы вызв ть предыдущую ком нду в конкретном экземпляре оболочки, н жмите кл вишу со стрелкой вверх. Просто, не пр вд ли? Продолж йте н жим ть стрелку вверх, чтобы вызыв ть более р нние ком нды в обр тном хронологическом порядке. Н жмите стрелку вниз, чтобы перейти к более поздней ком нде. Когд вы дойдете до нужной ком нды, н жмите Enter, чтобы з пустить ее.

Перемещение по истории ком нд является одним из двух н иболее популярных способов увеличения скорости р боты в ком ндной строке Linux (другой — это сопост вление имен ф йлов с ш блоном (*), которое мы изучили в гл ве 2). Перемещение эффективно, если нужн я ком нд н ходится нед леко в истории — не более двух или трех ш гов. Но кр йне утомительно добир ться этим способом до ком нд, которые вы вызыв ли д вно. Н жим ние стрелки быстро н доед ет.

Н илучший в ри нт использов ния перемещения по истории — это вызов и выполнение непосредственно предыдущей ком нды. Н многих кл ви тур х кл виш со стрелкой вверх н ходится рядом с кл вишей Enter, поэтому вы можете последов тельно н жим ть две кл виши быстрым движением п льцев. Н полнор змерной QWERTY-кл ви туре я кл ду безымянный п лец пр вой руки н стрелку вверх, ук з тельный п лец — н Enter, чтобы эффективно н жим ть обе кл виши (попробуйте с ми).

ЧАСТО ЗАДАВАЕМЫЕ ВОПРОСЫ ПРО ИСТОРИЮ КОМАНД

Сколько ком нд хр нится в истории?

М ксим льное число определяется зн чением переменной оболочки `HISTSIZE`, которую вы можете изменить:

```
$ echo $HISTSIZE
500
$ HISTSIZE=10000
```

Компьютерн я п мять н столько дешев и доступн , что имеет смысл уст новить большое зн чение `HISTSIZE`, чтобы вы могли вызыв ть и повторно з пуск ть ком нды из д лекого прошлого (история из 10 000 ком нд з ним ет всего около 200 Кб йт н жестком диске). Можно д же хр нить неогр ниченное количество ком нд, уст новив зн чение `-1`.

К кой текст доб вляется к истории ком нд?

Ком ндн я оболочк доб вляет в историю именно то, что вы вводите, без результ тов вычислений. Если вы з пустите `ls $HOME`, история будет содерж ть «ls $HOME», не «ls /home/smith» (исключение опис но во врезке «Выр жения, которые не отобр ж ются в истории ком нд» н с. 60).

Повторяющиеся ком нды доб вляются в историю?

В з висимости от зн чения переменной `HISTCONTROL`. По умолч нию, если эт переменн я не уст новлен , доб вляется к жд я ком нд . Если присвоить ей зн чение `ignoredups`

(что я рекомендую), то повторяющиеся ком нды не доб вляются, при условии что они идут одн з другой (другие зн чения см. в `man bash`):

```
$ HISTCONTROL=ignoredups
```

К ждый экземпляр оболочки имеет свою историю ком нд, или он общ я?

К ждый экземпляр интер ктивной оболочки имеет свою нез висимую от других историю ком нд.

Я з пустил новый экземпляр интер ктивной оболочки, и у него уже есть история. Почему?

Всякий р з, когд интер ктивн я оболочк з верш ет р боту, он з писыв ет свою историю в ф йл *$HOME/.bash_history* или другой, хр нящийся в переменной оболочки `HISTFILE`:

```
$ echo $HISTFILE
/home/smith/.bash_history
```

Новые экземпляры оболочки считыв ют при з пуске историю из этого ф йл . Если вы используете много оболочек, к жд я из них при выходе сохр няет свою историю в `$HISTFILE`, поэтому не всегд предск зуемо, к кую историю з грузит нов я оболочк .

Переменн я `HISTFILESIZE` определяет, сколько строк истории з писыв ется в ф йл. Если вы изменяете `HISTSIZE`, упр вляя р змером истории в п мяти, подум йте т кже об обновлении зн чения `HISTFILESIZE`:

```
$ echo $HISTFILESIZE
500
$ HISTFILESIZE=10000
```

Расширение истории команд

Р сширение истории — это функция оболочки, котор я обр щ ется к истории ком нд с помощью специ льных выр жений. Эти выр жения н чин ются с восклиц тельного зн к , который пользов тели Linux тр диционно н зыв ют *bang*. Дв восклиц тельных зн к подряд (*bang-bang*) озн ч ют предыдущую ком нду:

```
$ echo Efficient Linux
Efficient Linux
```

```
$ !!                          Bang bang = предыдущая команда
echo Efficient Linux          Оболочка выводит команду, которая будет запущена
Efficient Linux
```

Чтобы обратиться к последней команде, начинающейся с определенного выражения, поставьте перед ним восклицательный знак. Повторно запустить последнюю команду `grep` можно следующим образом:

```
$ !grep
grep Perl animals.txt
alpaca Intermediate Perl  2012  Schwartz, Randal
```

Чтобы обратиться к последней команде, которая содержала заданное выражение *где-либо*, не только в начале строки, окружите его вопросительными знаками[1]:

```
$ !?grep?
history | grep -w cd
 1000 cd $HOME/Music
 1092 cd ..
⋮
```

Вы также можете найти команду из истории оболочки по ее абсолютной позиции — идентификационному номеру слева от нее в выводе истории. Например, выражение `!1203` означает «команда на позиции 1023 в истории»:

```
$ history | grep hosts
 1203 cat /etc/hosts
$ !1203                       Команда под номером 1023
cat /etc/hosts
127.0.0.1    localhost
127.0.1.1    example.oreilly.com
::1          example.oreilly.com
```

Отрицательное значение извлекает команду из истории по ее относительной, не абсолютной позиции. Например, `!-3` означает «команда, которую вы выполнили три шага назад»:

```
$ history
 4197 cd /tmp/junk
 4198 rm *
 4199 head -n2 /etc/hosts
 4199 cd
 4200 history
```

[1] В этой команде можно убрать завершающий вопросительный знак: `!?grep`, но в некоторых случаях он необходим, например при расширении истории в стиле `sed` (см. раздел «Более эффективная замена с расширением истории»).

```
$ !-3                          Команда, запущенная три шага назад
head -n2 /etc/hosts
127.0.0.1    localhost
127.0.1.1    example.oreilly.com
```

Р сширение истории быстро и удобно в использов нии. Одн ко оно может приводить к ошибк м, если вы ук жете непр вильное зн чение. Посмотрите вним тельно н предыдущий пример. Если бы вы просчит лись и н бр ли `!-4` вместо `!-3`, то з пустили бы `rm *` вместо предпол г емой ком нды `head` и по ошибке уд лили бы все ф йлы в дом шнем к т логе! Для снижения рисков доб вьте модифик тор `:p`, чтобы н печ т ть ком нду из в шей истории, но не выполнять ее:

```
$ !-3:p
head -n2 /etc/hosts            Вывести на экран, но не исполнять
```

Оболочк доб вляет невыполненную ком нду (`head`) в историю. Т ким обр зом, если вы видите, что все в порядке, можно быстро з пустить ее с помощью *bang-bang*:

```
$ !-3:p
head -n2 /etc/hosts            Вывести на экран, не исполнять и записать в историю
$ !!                           Запустить команду
head -n2 /etc/hosts            Выведена на экран и исполнена
127.0.0.1 localhost
127.0.1.1 example.oreilly.com
```

Р сширение истории иногд н зыв ют *bang-ком нд ми*, но выр жения вроде `!!` и `!grep` не являются ком нд ми. Это строковые выр жения, которые вы можете поместить в любом месте ком нды. Используйте ком нду `echo` для вывод зн чения `!!` в stdout, не выполняя его, и подсчит йте количество слов с помощью `wc`:

```
$ ls -l /etc | head -n3       Запустим любую команду
total 1584
drwxr-xr-x 2 root     root     4096 Jun 16 06:14 ImageMagick-6/
drwxr-xr-x 7 root     root     4096 Mar 19 2020 NetworkManager/

$ echo "!!" | wc -w           Подсчитаем количество слов в предыдущей команде
echo "ls -l /etc | head -n3" | wc -w
6
```

Этот пример демонстрирует, что р сширения истории могут использов ться не только для повторного выполнения ком нд. В следующем р зделе вы позн комитесь с другими способ ми их применения. Для получения полной информ ции об особенностях истории ком нд з пустите `man history`.

Выражения, которые не отображаются в истории команд

Оболочк доб вляет ком нды в историю без вычислений, к к ук з но во врезке «Ч сто з д в емые вопросы про историю ком нд» н с. 56. Единственным исключением из этого пр вил является р сширение истории. Его выр жения всегд вычисляются перед доб влением в историю ком нд:

```
$ ls                  Запустим любую команду
hello.txt
$ cd Music            Запустим другую команду
$ !-2                 Используем расширение истории
ls
song.mp3
$ history             Посмотрим историю
 1000 ls
 1001 cd Music
 1002 ls              «ls» записана в историю вместо «!-2»
 1003 history
```

Это пр вило имеет смысл. Предст вьте, что вы пыт етесь понять историю ком нд, полную т ких выр жений, к к `!-15` и `!-92`, которые ссыл ются н другие з писи в истории. Возможно, в м придется проследить путь через всю историю, чтобы понять всего лишь одну ком нду.

Забудьте об ошибочном удалении файлов (спасибо расширению истории)

Случ лось ли т кое, что вы хотели уд лить ф йлы с помощью ш блон , н пример `*.txt`, но из-з ошибки в н боре стерли не те ф йлы? Вот пример с пробелом после звездочки:

```
$ ls
123 a.txt    b.txt    c.txt    dont-delete-me    important-file    passwords
$ rm * .txt                    ОПАСНО! Не запускайте это!
```

Популярным способом избеж ть этой оп сности является псевдоним `rm`, з пуск ющий ком нду `rm -i`, котор я з пр шив ет подтверждение перед к ждым уд лением:

```
$ alias rm='rm -i'             Часто уже есть в конфигурационном файле оболочки
$ rm *.txt
/bin/rm: remove regular file 'a.txt'? y
/bin/rm: remove regular file 'b.txt'? y
/bin/rm: remove regular file 'c.txt'? y
```

В результ те лишний пробел не будет ф т льным, потому что подск зки от `rm -i` помогут понять, что вы уд ляете не те ф йлы:

```
$ rm * .txt
/bin/rm: remove regular file '123'?              Что-то не так: отменить команду
```

Одн ко решение с псевдонимом не всегд эффективно, потому что в большинстве случ ев з прос подтверждения при уд лении ф йл не нужен и только р здр ж ет. Кроме того, оно не ср бот ет, если вы войдете в систему н другом компьютере с Linux, где т кой псевдоним не н строен.

Есть лучший способ избеж ть ошибочного выполнения сопост вления имен ф йлов с ш блоном. Этот метод состоит из двух эт пов и основ н н р сширении истории:

1. *Проверк* . Перед з пуском `rm` з пустите `ls` с нужным ш блоном, чтобы увидеть, к кие ф йлы совп д ют.

   ```
   $ ls *.txt
   a.txt b.txt c.txt
   ```

2. *Уд ление*. Если вывод `ls` прошел проверку, з пустите `rm !$`, чтобы уд лить те же ф йлы, которые были сопост влены с ш блоном[1].

   ```
   $ rm !$
   rm *.txt
   ```

Р сширение истории `!$` озн ч ет «последнее слово, которое вы ввели в предыдущей ком нде». Т ким обр зом, `rm !$` здесь является сокр щением для «уд лить все, что я только что перечислил с помощью `ls`», именно `*.txt`. Если вы случ йно доб вите пробел после звездочки, вывод `ls` предупредит, что что-то пошло не т к:

```
$ ls * .txt
/bin/ls: cannot access '.txt': No such file or directory
123  a.txt    b.txt    c.txt    dont-delete-me    important-file    passwords
```

К к здорово, что вы сн ч л з пустили `ls`, не `rm`! Теперь вы можете изменить ком нду, уд лив лишний пробел, и продолжить без потери д нных. Опис нн я последов тельность из ком нд `ls` и `rm !$` — отличный прием, который з служив ет включения в инструмент рий р боты с Linux.

---

[1] Предпол г ется, что после выполнения `ls` без в шего ведом не были доб влены или уд лены ф йлы, соответствующие ш блону. Не пол г йтесь н этот способ при р боте с к т лог ми, содержимое которых быстро изменяется.

Похожий метод з ключ ется в просмотре содержимого ф йл с помощью ком нды `head` перед его уд лением. Убедившись, что выбр ли пр вильный ф йл, можете з тем з пустить `rm !$`:

```
$ head myfile.txt
(отображаются первые 10 строк файла)
$ rm !$
rm myfile.txt
```

Т кже в оболочке имеется р сширение истории `!*`, которое соответствует всем ргумент м, введенным в ми в предыдущей ком нде, не только последнему:

```
$ ls *.txt *.o *.log
a.txt b.txt c.txt main.o output.log parser.o
$ rm !*
rm *.txt *.o *.log
```

Н пр ктике я использую `!*` зн чительно реже, чем `!$`, т к к к есть риск, что в случ е ошибки звездочк * будет интерпретиров н к к символ ш блон для имен ф йлов. Поэтому применение `!*` не н много безоп снее, чем н бор ш блон , н пример `*.txt`.

Инкрементальный поиск по истории команд

Было бы здорово, если после ввод нескольких символов ком нды ост льные появлялись бы мгновенно — и ком нд готов к з пуску! И это возможно. Быстр я функция оболочки, н зыв ем *я инкремент льным поиском*, похож н интер ктивные подск зки поисковых систем. В большинстве случ ев инкремент льный поиск — это с мый простой и быстрый способ вспомнить ком нды из истории, д же те, которые вы з пуск ли д вно. Я н стоятельно рекомендую доб вить инкремент льный поиск в свой н бор инструментов Linux:

1. В пригл шении оболочки н жмите Ctrl-R (R озн ч ет обр тный (reverse) инкремент льный поиск).
2. Вводите любую ч сть р нее з пущенной ком нды — н ч ло, середину или конец.
3. С к ждым введенным символом оболочк н ходит ближ йшую в истории ком нду, котор я соответствует в шему вводу.
4. Когд вы увидите нужную ком нду, н жмите Enter, чтобы з пустить ее.

Предположим, вы нед вно з пуск ли ком нду `cd $HOME/Finances/Bank` и хотите повторить ее. Н жмите Ctrl-R в ком ндной строке. Подск зк изменится, ук зыв я н инкремент льный поиск:

```
(reverse-i-search)`':
```

Н чните вводить нужную ком нду. Н пример, введите `c`:

```
(reverse-i-search)`': c
```

Оболочк отобр ж ет с мую последнюю ком нду, содерж щую символ `c`, выделяя то, что вы н бр ли:

```
(reverse-i-search)`': less /etc/hosts
```

Введите следующий символ, `d`:

```
(reverse-i-search)`': cd
```

Оболочк отобр ж ет с мую последнюю ком нду, содерж щую `cd`, снов выделяя то, что вы н бр ли:

```
(reverse-i-search)`': cd /usr/local
```

Продолж йте вводить ком нду, доб вляя пробел и зн к долл р :

```
(reverse-i-search)`': cd $
```

Ком ндн я строк приобрет ет следующий вид:

```
(reverse-i-search)`': cd $HOME/Finances/Bank
```

Это ком нд , которую вы иск ли, поэтому н жмите Enter, чтобы з пустить ее. Результ т получен всего лишь з пять н ж тий кл виш.

Здесь я предположил, что `cd $HOME/Finances/Bank` был последней соответствующей з просу ком ндой в истории. А что, если это не т к? Если вы ввели множество ком нд, содерж щих одну и ту же строку, то предыдущий инкремент льный поиск пок з л бы другое совп дение, н пример:

```
(reverse-i-search)`': cd $HOME/Music
```

Что теперь? Вы можете ввести больше символов, чтобы конкретизиров ть ком нду, но лучше н жмите Ctrl-R еще р з. Это н ж тие кл виши з ст вляет оболочку перейти к *следующей* соответствующей ш блону поиск ком нде в истории:

```
(reverse-i-search)`': cd $HOME/Linux/Books
```

Продолж йте н жим ть Ctrl-R, пок не дойдете до нужной ком нды:

```
(reverse-i-search)`': cd $HOME/Finances/Bank
```

и н жмите Enter для з пуск .

Вот еще несколько трюков с инкремент льным поиском:

- Чтобы вызв ть последнюю строку, которую вы иск ли и выполняли, н чните с н ж тия Ctrl-R дв р з подряд.
- Чтобы ост новить инкремент льный поиск и продолжить р боту н д текущей ком ндой, н жмите Escape, Ctrl-J или любую кл вишу для ред ктиров ния в ком ндной строке (см. следующий р здел в этой гл ве), н пример кл вишу со стрелкой влево или впр во.
- Чтобы выйти из инкремент льного поиск и очистить ком ндную строку, н жмите Ctrl-G или Ctrl-C.

Не ж лейте времени н тренировку н выков инкремент льного поиск . Вскоре вы будете н ходить ком нды с невероятной скоростью[1].

Редактирование командной строки

Есть множество причин для ред ктиров ния ком нды во время ее ввод или после того, к к вы ее з пустили:

- Чтобы испр вить ошибки.
- Чтобы сконструиров ть ком нду по ч стям. Н пример, сн ч л н бр в конец ком нды, з тем перейдя к н ч лу строки и н бр в н ч ло.
- Для созд ния новой ком нды н основе предыдущей из истории ком нд (к к вы увидите в гл ве 8, это ключевой н вык для созд ния сложных конвейеров).

В этом р зделе я пок жу в м три метод , которые позволят улучшить н выки ред ктиров ния ком нд и повысить скорость р боты:

Перемещение курсор

С мый медленный и н именее мощный метод, но простой в освоении.

Использов ние зн к вст вки (caret)

Один из видов р сширения истории.

Н ж тия кл виш в стиле Emacs или Vim

М ксим льно эффективное ред ктиров ние ком ндной строки.

Изучите все три метод для большей гибкости в ред ктиров нии ком ндной строки.

---

[1] При н пис нии этой книги я ч сто перез пуск л ком нды системы контроля версий, т кие к к `git add`, `git commit` и `git push`. Инкремент льный поиск упростил повторный з пуск этих ком нд.

Перемещение курсора внутри команды

Н жим йте кл виши со стрелк ми влево и впр во для перемещения н один символ вперед и н з д по ком ндной строке. Используйте кл виши Backspace или Delete, чтобы уд лять текст. Т блиц 3.1 обобщ ет ст нд ртные комбин ции кл виш для ред ктиров ния ком ндной строки.

Перемещ ться вперед и н з д легко, но неэффективно. Этот метод подходит, когд необходимо внести небольшие и простые изменения.

Таблица 3.1. Клавиши перемещения курсора для простого редактирования в командной строке

Сочетание клавиш	Действие
Стрелка влево	Перейти влево на один символ
Стрелка вправо	Перейти вправо на один символ
Ctrl + стрелка влево	Перейти влево на одно слово
Ctrl + стрелка вправо	Перейти вправо на одно слово
Home	Перейти к началу командной строки
End	Перейти к концу командной строки
Backspace	Удалить один символ перед курсором
Delete	Удалить один символ под курсором

Расширение истории с помощью знака вставки

Предположим, вы по ошибке з пустили следующую ком нду, н бр в `jg` вместо `jpg`:

```
$ md5sum *.jg | cut -c1-32 | sort | uniq -c | sort -nr
md5sum: '*.jg': No such file or directory
```

Чтобы выполнить ком нду пр вильно, вы можете вызв ть ее из истории, переместить курсор и испр вить ошибку, но есть более быстрый способ достичь цели. Просто введите ст рый (непр вильный) текст, новый (испр вленный) текст и п ру зн ков вст вки (^), н пример:

```
$ ^jg^jpg
```

Н жмите Enter — появится и ср зу же будет з пущен пр вильн я ком нд :

```
$ ^jg^jpg
md5sum *.jpg | cut -c1-32 | sort | uniq -c | sort -nr
⋮
```

Синт ксис со зн ком вст вки, который является видом р сширения истории, озн ч ет: «В предыдущей ком нде вместо `jg` подст вьте `jpg`». Обр тите вним - ние, что оболочк выводит новую ком нду перед ее выполнением, что является ст нд ртным поведением для р сширения истории.

Этот метод изменяет только первое вхождение исходного код (`jg`). Если в ш исходн я ком нд содерж л `jg` более одного р з , только первый экземпляр изменится н `jpg`.

БОЛЕЕ МОЩНАЯ ЗАМЕНА С ПОМОЩЬЮ РАСШИРЕНИЯ ИСТОРИИ

Возможно, вы зн комы с использов нием ком нд `sed` или `ed` для преобр зов ния исходной строки в необходимую в м:

```
s/source/target/
```

Ком ндн я оболочк поддержив ет н логичный синт ксис. Н чните с выр жения для р сширения истории, чтобы вызв ть ком нду, н пример `!!`. З тем доб вьте двоеточие и з кончите з меной в стиле `sed`. Н пример, чтобы вызв ть предыдущую ком нду и з менить `jg` н `jpg` (только при первом вхождении) з пустите:

```
$ !!:s/jg/jpg/
```

Вы можете н ч ть с любого р сширения истории, которое в м нр вится. Н пример, `!md5sum` вызовет последнюю ком нду, н чин ющуюся с `md5sum`, и выполнит ту же з мену `jg` н `jpg`:

```
$ !md5sum:s/jg/jpg/
```

Этот способ может пок з ться сложным, но иногд он позволяет быстрее достичь цели, чем другие методы ред ктиров ния в ком ндной строке. З пустите `man history` для получения полной информ ции.

Редактирование командной строки в стилях Emacs или Vim

С мый эффективный способ ред ктиров ния ком ндной строки — зн комые сочет ния кл виш, вдохновленные текстовыми ред ктор ми Emacs и Vim. Если вы уже имеете опыт р боты с одним из этих ред кторов, то можете ср зу перейти к этому стилю ред ктиров ния ком ндной строки. Если нет, т бл. 3.2 поможет в м н ч ть р боту с н иболее р спростр ненными сочет ниями кл виш для перемещения и ред ктиров ния. Обр тите вним ние, что кл виш Meta в Emacs обычно з меняется н Escape (н ж т и отпущен) или Alt (н ж т и удержив ется).

По умолч нию в оболочке используется ред ктиров ние в стиле Emacs, и я рекомендую его к к более простой в освоении и использов нии. Если вы предпочит ете ред ктиров ние в стиле Vim, выполните следующую ком нду (или доб вьте ее в свой ф йл *$HOME/.bashrc* и з грузите):

```
$ set -o vi
```

Н жмите Escape для вход в режим ред ктиров ния ком нды, з тем используйте кл виши из столбц Vim в т бл. 3.2. Чтобы вернуться к ред ктиров нию в стиле Emacs, з пустите:

```
$ set -o emacs
```

А теперь пр ктик , пр ктик и еще р з пр ктик , пок сочет ния кл виш не ст нут в шей второй н турой. Поверьте, потр ченное время быстро окупится.

Таблица 3.2. Сочетания клавиш для редактирования в стилях Emacs и Vim[1]

Действие	Emacs	Vim
Перейти вперед на один символ	Ctrl-f	h
Перейти назад на один символ	Ctrl-b	l
Перейти вперед на одно слово	Meta-f	w
Перейти назад на одно слово	Meta-b	b
Перейти в начало строки	Ctrl-a	0
Перейти в конец строки	Ctrl-e	$
Поменять местами два символа	Ctrl-t	xp

[1] Действия, помеченные к к *n/a*, не выполняются простым сочет нием кл виш, но могут выполняться с помощью более длинных последов тельностей.

Действие	Emacs	Vim
Поменять местами два слова	Meta-t	*n/a*
Сделать заглавной первую букву следующего слова	Meta-c	w~
Изменить регистр следующего слова на верхний	Meta-u	*n/a*
Изменить регистр следующего слова на нижний	Meta-l	*n/a*
Изменить регистр текущего символа	*n/a*	~
Вставить следующий символ (в т. ч. управляющие символы)	Ctrl-v	Ctrl-v
Удалить один символ перед курсором	Ctrl-d	x
Удалить один символ после курсора	Backspace или Ctrl-h	X
Удалить одно слово перед курсором	Meta-d	dw
Удалить одно слово после курсора	Meta-Backspace или Ctrl-w	db
Удалить от курсора до начала строки	Ctrl-u	d^
Удалить от курсора до конца строки	Ctrl-k	D
Удалить строку целиком	Ctrl-e Ctrl-u	dd
Вставить (скопировать) последний удаленный текст	Ctrl-y	p
Вставить (скопировать) предпоследний удаленный текст	Meta-y	*n/a*
Отменить предыдущую операцию редактирования	Ctrl-_	u
Отменить все изменения	Meta-r	U
Переключение из режима вставки в командный режим	*n/a*	Escape
Переключение из командного режима в режим вставки	*n/a*	i
Прервать текущую операцию редактирования	Ctrl-g	*n/a*
Очистить экран	Ctrl-l	Ctrl-l

Дополнительные сведения о редактировании в стиле Emacs см. в разделе *Bindable Readline Commands*[1] в руководстве *GNU* по `bash`. О редактировании в стиле Vim см. документ *Readline VI Editing Mode Cheat Sheet*[2].

[1] https://www.gnu.org/software/bash/manual/html_node/Bindable-Readline-Commands.html.

[2] https://catonmat.net/ftp/bash-vi-editing-mode-cheat-sheet.pdf.

Резюме

Три техники упрощ ют использов ние Linux:

- Безоп сное уд ление ф йлов с помощью `!$`.
- Инкремент льный поиск через Ctrl-R.
- Ред ктиров ние ком ндной строки в стиле Emacs.

Пр ктикуйтесь в прием х, опис нных в этой гл ве, и вы зн чительно ускорите р боту с ком ндной строкой.

ГЛАВА 4

Перемещение по файловой системе

В фильме «Приключения Б к ру Б нз я в восьмом измерении», кл ссической мерик нской комедии 1984 год , лихой гл вный герой произносит следующие мудрые слов в духе дзен: «Помни, куд бы ты ни пошел... ты здесь». Б к ру вполне мог говорить о ф йловой системе Linux:

```
$ cd /usr/share/lib/etc/bin          Куда бы ты ни пошел
$ pwd
/usr/share/lib/etc/bin               ...ты здесь.
```

Другими слов ми, где бы вы ни н ходились в ф йловой системе Linux, в итоге все р вно перейдете из текущего к т лог куд -то еще (в другой к т лог). Чем быстрее и эффективнее вы сможете выполнять эту н виг цию, тем продуктивнее вы будете.

Приемы, опис нные в этой гл ве, помогут в м быстрее ориентиров ться в ф йловой системе, сокр щ я количество опер ций ввод . Они выглядят обм нчиво простыми, но приносят *огромную* пользу. Эти методы делятся н две большие к тегории:

- Быстрое перемещение в конкретный к т лог.
- Быстрое возвр щение в к т лог, который вы посещ ли р нее.

Чтобы освежить в п мяти в ши зн ния о к т лог х Linux, см. Приложение А. Если вы используете оболочку, отличную от `bash`, то н йдете дополнительную информ цию в Приложении Б.

Лучшие способы перехода в нужный каталог

Если вы спросите десять опытных пользов телей Linux, что с мое утомительное в ком ндной строке, не менее семи из них ответят: «Ввод длинных путей к к т лог м»[1]. В конце концов, если в ши р бочие ф йлы н ходятся в */home/smith/Work/Projects/Apps/Neutron-Star/src/include*, фин нсовые документы леж т в */home/smith/Finances/Bank/Checking/Statements*, в ши видео — в */data/Arts/Video/Collection*, утомительно вводить эти пути снов и снов . В этом р зделе вы узн ете, к к удобнее всего перейти к з д нному к т логу.

Переход в домашний каталог

Н чнем с основ. Нез висимо от того, где вы н ходитесь в ф йловой системе, вы можете вернуться в свой дом шний к т лог, з пустив `cd` без ргументов:

```
$ pwd
/etc                    Вы начали где-то в файловой системе
$ cd                    Запустите cd без аргументов...
$ pwd
/home/smith             ...и вы снова в домашнем каталоге
```

Чтобы перейти к подк т лог м в в шем дом шнем к т логе из любой точки ф йловой системы, обр титесь к дом шнему к т логу с помощью сокр щения. Одним из сокр щений является переменн я оболочки `HOME`:

```
$ cd $HOME/Work
```

Еще один способ — это символ тильд (~):

```
$ cd ~/Work
```

И `$HOME`, и ~ являются выр жениями, вычисляемыми оболочкой. Это можно проверить с помощью ст нд ртного вывод :

```
$ echo $HOME ~
/home/smith /home/smith
```

Тильд т кже может привести к дом шнему к т логу другого пользов теля, если помещен непосредственно перед его именем:

```
$ echo ~jones
/home/jones
```

[1] Я не проводил опрос, но, безусловно, это вполне ре листичный в ри нт.

Перемещайтесь быстрее с автозавершением командной строки

Когд вы вводите ком нду cd, не печ т йте путь полностью. Вместо этого н жмите кл вишу Tab, чтобы втом тически подст влять имен к т логов. В к честве тренировки посетите к т лог, содерж щий подк т логи, н пример */usr*:

```
$ cd /usr
$ ls
bin games include lib local sbin share src
```

Предположим, вы хотите перейти в подк т лог *share*. Введите sha и один р з н жмите Tab:

```
$ cd sha<Tab>
```

Ком ндн я оболочк дополнит имя к т лог з в с:

```
$ cd share/
```

Это полезное сокр щение н зыв ется *втоз вершением ком ндной строки* (*tab completion*). Оно ср б тыв ет ср зу, когд введенный в ми текст приходит в соответствие с именем одного из к т логов. Когд текст соответствует имен м нескольких к т логов, в шей оболочке требуется дополнительн я информ ция. Предположим, вы н бр ли только s и н ж ли Tab:

```
$ cd s<Tab>
```

Оболочк не может з вершить имя к т лог *share* (пок что), потому что имен других к т логов тоже н чин ются с s: *sbin* и *src*. Н жмите Tab второй р з, и оболочк выведет все возможные в ри нты з вершения, чтобы помочь в м:

```
$ cd s<Tab><Tab>
sbin/ share/ src/
```

Теперь он будет ожид ть в шего следующего действия. Чтобы устр нить двусмысленность, введите символ h и еще р з н жмите кл вишу Tab:

```
$ cd share/
```

В общем, н жим йте Tab один р з, когд ост лся единственно возможный в ри нт, и дв жды, чтобы вывести все возможные в ри нты. Чем больше символов вы вводите, тем меньше двусмысленности.

Автоз вершение с помощью кл виши Tab отлично подходит для ускорения н виг ции. Вместо того чтобы вводить длинный путь, т кой к к */home/smith/*

Projects/Web/src/include, введите столько символов, сколько считает нужным, и продолжайте нажимать клавишу Tab. Попрактикуйтесь, и вы быстро освоитесь.

Автозавершение зависит от программы

Автозавершение командной строки подходит не только для команд `cd`. Оно работает для большинства команд, хотя его поведение может отличаться. С командой `cd` клавиша Tab завершает имена каталогов. Для команд, работающих с файлами, таких как `cat`, `grep` и `sort`, автозавершение дополняет имена файлов. Если используется команда `ssh` (*secure shell*), Tab дополняет имена хостов. Для команды `chown` (изменить владельца файла) он дополняет имена пользователей. Вы можете даже создать свои собственные правила завершения для увеличения скорости работы, как мы увидим далее в листинге 4.1. Также вы можете запустить `man bash` и прочитать раздел *Programmable completion*.

Переход к часто посещаемым каталогам с использованием псевдонимов или переменных

Если вы часто посещаете удаленный от корня файловой системы каталог, такой, например, как */home/smith/Work/Projects/Web/src/include*, создайте псевдоним, выполняющий операцию `cd`:

```
# В файле конфигурации оболочки:
alias work="cd $HOME/Work/Projects/Web/src/include"
```

Теперь запустите псевдоним, чтобы добраться до нужного каталога:

```
$ work
$ pwd
/home/smith/Work/Projects/Web/src/include
```

В качестве альтернативы можно создать переменную для хранения пути к нужному каталогу:

```
$ work=$HOME/Work/Projects/Web/src/include
$ cd $work
$ pwd
/home/smith/Work/Projects/Web/src/include
$ ls $work/css                    Использование переменной другими способами
main.css mobile.css
```

Редактируйте часто используемые файлы с помощью псевдонимов

Иногд причиной ч стого посещения к т лог является ред ктиров ние определенного ф йл . Тогд есть смысл определить псевдоним для ред ктиров ния этого ф йл . Определение псевдоним `rcedit` в примере ниже позволит в м ред ктиров ть *$HOME/.bashrc* нез висимо от того, где вы н ходитесь в ф йловой системе. З пуск ком нды `cd` не требуется:

```
# Поместите в файл конфигурации оболочки и примените его:
alias rcedit='$EDITOR $HOME/.bashrc'
```

Если вы регулярно посещ ете множество к т логов с длинными путями, можете созд ть псевдонимы или переменные для к ждого из них. Одн ко этот подход имеет недост тки:

- Трудно з помнить все эти псевдонимы и переменные.
- Вы можете случ йно созд ть псевдоним с тем же именем, что и у существующей ком нды, и это вызовет конфликт.

В к честве льтерн тивы можно созд ть функцию оболочки, подобную `qcd` (*quick cd*) в листинге 4.1. Эт функция приним ет строковый ключ в к честве ргумент , н пример `work` или `recipes`, и з пуск ет ком нду `cd` для выбр нного пути к к т логу.

Листинг 4.1. Функция для перехода в отдаленные каталоги

```
# Определение функции qsd
qcd () {
  # Принимает 1 аргумент, который является строковым ключом
  case "$1" in
    work)
      cd $HOME/Work/Projects/Web/src/include
      ;;
    recipes)
      cd $HOME/Family/Cooking/Recipes
      ;;
    video)
      cd /data/Arts/Video/Collection
      ;;
    beatles)
      cd $HOME/Music/mp3/Artists/B/Beatles
      ;;
    *)
      # Введенный аргумент не совпал ни с одним из ключей
      echo "qcd: unknown key '$1'"
      return 1
      ;;
```

```
  esac
  # Вывести на экран имя текущего каталога, чтобы указать, где вы находитесь
  pwd
}
# Настройка автозавершения
complete -W "work recipes video beatles" qcd
```

Сохр ните функцию в ф йле конфигур ции оболочки, т ком к к *$HOME/.bashrc* (см. р здел «Окружения и ф йлы иници лиз ции, кр тк я версия» н с. 50), обновите экземпляр оболочки — и он готов к з пуску. Введите `qcd`, з тем один из поддержив емых ключей, чтобы быстро перейти в соответствующий к т лог:

```
$ qcd beatles
/home/smith/Music/mp3/Artists/B/Beatles
```

В к честве бонус последняя строк сцен рия з пуск ет встроенную в оболочку ком нду `complete`, котор я н стр ив ет втоз вершение для ком нды `qcd`. Поэтому сцен рий з верш ется четырьмя поддержив емыми ключ ми. Теперь в м не нужно з помин ть ргументы `qcd`! Просто введите `qcd`, пробел, дв жды н жмите кл вишу Tab — и оболочк н печ т ет все доступные ргументы. Тогд вы сможете ввести любой из них обычным способом:

```
$ qcd <Tab><Tab>
beatles recipes video work
$ qcd v<Tab><Enter>                    Завершает 'v' до 'video'
/data/Arts/Video/Collection
```

Уменьшите пространство поиска с помощью CDPATH

Функция `qcd` обр б тыв ет только ук з нные в ми к т логи. Но ком ндн я оболочк предост вляет более общее решение для созд ния быстрых функций по переходу в нужный к т лог, которое лишено этого недост тк . *Путь поиск ком нды cd* (*cd search path*) в свое время изменил мой подход к н виг ции по ф йловой системе Linux.

Предположим, у в с есть в жный подк т лог */home/smith/Family/Memories/Photos*, который вы ч сто посещ ете. Когд вы перемещ етесь по ф йловой системе, чтобы поп сть в к т лог *Photos*, в м может потребов ться ввести длинный путь, н пример:

```
$ cd ~/Family/Memories/Photos
```

Было бы здорово, если бы вы могли сокр тить этот путь до *Photos*, нез висимо от того, где вы н ходитесь в ф йловой системе.

```
$ cd Photos
```

Обычно эт ком нд не р бот ет:

```
bash: cd: Photos: No such file or directory      Нет такого файла или каталога
```

Ср бот ет он , если вы случ йно ок з лись в исходном к т логе (*~/Family/Memories*) или в к ком-либо другом к т логе с подк т логом *Photos*. Но путем несложных м нипуляций вы можете ук з ть ком нде `cd` иск ть подк т лог *Photos* не только в текущем к т логе. Поиск р бот ет пр ктически мгновенно и только в ук з нных в ми родительских к т лог х. Н пример, вы можете ук з ть `cd` иск ть в *$HOME/Family/Memories* в дополнение к текущему к т логу. После этого `cd Photos` з вершится успешно из любой точки ф йловой системы:

```
$ pwd
/etc
$ cd Photos
/home/smith/Family/Memories/Photos
```

Путь поиск ком нды `cd` р бот ет т к же, к к путь `$PATH`, но вместо ком нд он н ходит подк т логи. Н стройте его с помощью переменной оболочки `CDPATH`, котор я имеет тот же форм т, что и `PATH` — список к т логов, р зделенных двоеточиями. Н пример, в ш переменн я `CDPATH` состоит из четырех к т логов:

```
$HOME:$HOME/Projects:$HOME/Family/Memories:/usr/local
```

и вы печ т ете:

```
$ cd Photos
```

тогд ком нд `cd` проверит н личие следующих к т логов по порядку:

1. Подк т лог *Photos* в текущем к т логе
2. *$HOME/Photos*
3. *$HOME/Projects/Photos*
4. *$HOME/Family/Memories/Photos*
5. */usr/local/Photos*

В н шем случ е `cd` с четвертой попытки успешно меняет к т лог н *$HOME/Family/Memories/Photos*. Если дв к т лог в `$CDPATH` имеют подк т лог с именем *Photos*, переход произойдет в тот из них, который стоит р ньше в списке.

Обычно успешно выполненн я ком нд `cd` не выводит ник ких результ тов. Одн ко, когд `cd` н ходит к т лог, используя `CDPATH`, он выводит бсолютный путь, чтобы сообщить о новом текущем к т логе:

```
$ CDPATH=/usr          Установить CDPATH
$ cd /tmp              Нет вывода: CDPATH не использовался
$ cd bin               cd использовала CDPATH...
/usr/bin               ...и вывела путь к новому рабочему каталогу
```

З полните CDPATH н иболее в жными и ч сто используемыми родительскими к т лог ми, и вы сможете перейти в любой их подк т лог из любой точки ф йловой системы, не вводя большую ч сть пути. Поверьте мне, это *потряс юще*, и следующий пример док жет это.

Организуйте свой домашний каталог для быстрой навигации

Д в йте воспользуемся переменной CDPATH, чтобы упростить н виг цию по дом шнему к т логу. После короткой н стройки к т логи в в шей дом шней директории ст нут легкодоступными. Нез висимо от того, где вы н ходитесь в ф йловой системе, потребуется миним льный н бор текст . Этот метод особенно эффективен, если в ш дом шний к т лог хорошо орг низов н и содержит не более двух уровней подк т логов. Н рис. 4.1 пок з н пример хорошо орг - низов нного дом шнего к т лог .

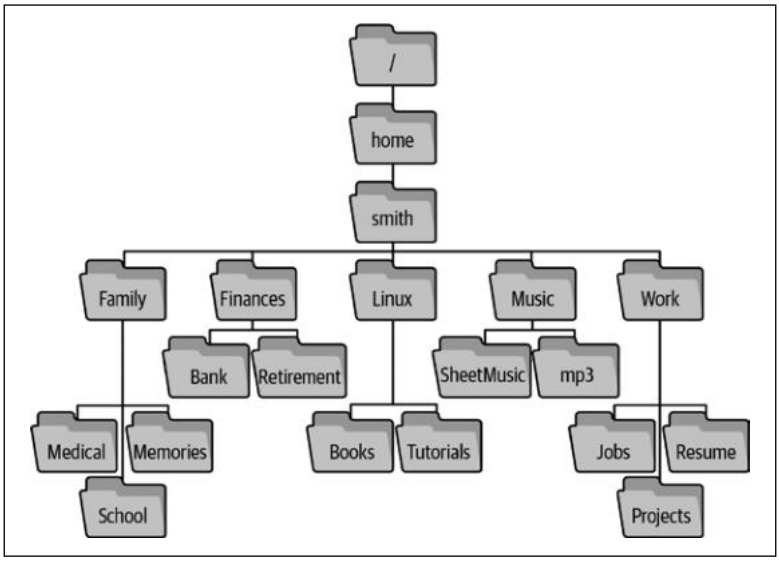


Рис. 4.1. Два уровня подкаталогов в каталоге */home/smith*

Хитрость з ключ ется в том, чтобы в ш CDPATH включ л следующее:

1. $HOME.
2. В ш н бор подк т логов $HOME.
3. Относительный путь к родительскому к т логу, обозн ченный двумя точк ми (..).

Включив $HOME в CDPATH, вы можете ср зу перейти к любому из его подк т логов (*Family, Finances, Linux, Music* и *Work*) из любого мест в ф йловой системе, не вводя н ч льный путь:

```
$ pwd
/etc                          Начните за пределами вашего домашнего каталога
$ cd Work
/home/smith/Work
$ cd Family/School            Вы перешли на один уровень ниже $HOME
/home/smith/Family/School
```

Включив подк т логи $HOME в CDPATH, вы сможете ср зу перейти в их подк т логи:

```
$ pwd
/etc                          Где-то за пределами вашего домашнего каталога
$ cd School
/home/smith/Family/School     Вы перешли на два уровня ниже $HOME
```

Все к т логи в в шей переменной CDPATH до сих пор являлись бсолютными путями в $HOME и его подк т лог х. Одн ко, включив относительный путь .., вы поменяете поведение ком нды cd. Нез висимо от того, где вы н ходитесь в ф йловой системе, вы сможете перейти к любому к т логу того же уровня по его имени. Н пример, если вы н ходитесь в */usr/bin* и хотите перейти в */usr/lib*:

```
$ pwd
/usr/bin                      Ваш текущий каталог
$ ls ..
bin include lib src           Ваши одноуровневые каталоги
$ cd lib
/usr/lib                      Вы перешли в другой одноуровневый каталог
```

Или, если вы прогр ммист, р бот ющий с подк т лог ми *src*, *include* и *docs*:

```
$ pwd
/usr/src/myproject
$ ls
docs include src
```

вы можете быстро перемещ ться между подк т лог ми:

```
$ cd docs                     Измените ваш текущий каталог
$ cd include
/usr/src/myproject/include    Вы перешли в другой одноуровневый каталог
$ cd src
/usr/src/myproject/src        Снова перешли
```

CDPATH для дерев к т логов н рис. 4.1 может содерж ть шесть элементов: в ш дом шний к т лог, четыре его подк т лог и относительный путь к родительскому к т логу:

```
# Поместите в файл настроек оболочки и примените его
export CDPATH=$HOME:$HOME/Work:$HOME/Family:$HOME/Linux:$HOME/Music:..
```

После применения ф йл конфигур ции вы сможете перейти к в жным к т лог м, вводя только их короткие имен . Ур !

Этот метод р бот ет лучше всего, если все подк т логи в к т лог х, ук з нных в CDPATH, имеют уник льные имен . Если у в с есть повторяющиеся имен , н пример *$HOME/Music* и *$HOME/Linux/Music*, то ком нд `cd Music` всегд будет проверять *$HOME* перед *$HOME/Linux* и, следов тельно, не н йдет *$HOME/Linux/Music*.

Чтобы проверить н личие повторяющихся имен подк т логов н первых двух уровнях `$HOME`, попробуйте однострочник, который перечисляет все к т логи и подк т логи `$HOME`, изолирует имен подк т логов с помощью `cut`, сортирует список и подсчитыв ет вхождения с помощью `uniq`:

```
$ cd
$ ls -d */ && (ls -d */*/ | cut -d/ -f2-) | sort | uniq -c | sort -nr | less
```

Вы могли узн ть ком нду из р здел «Обн ружение дублик тов ф йлов» н с. 32. Если в выходных д нных отобр ж ется число больше 1, у в с есть дублик ты. Эт ком нд включ ет в себя несколько функций, которые мы еще не р ссм трив ли. Вы позн комитесь с двойным мперс ндом (`&&`) в р зделе «Способ 1: Условные списки» н с. 135, со скобк ми — в р зделе «Способ 10: Явные подоболочки» н с. 159.

Лучшие способы вернуться в каталог

Вы только что н учились эффективно переходить в нужные к т логи. Теперь я пок жу в м, к к быстро снов перейти в к т лог, который вы уже посещ ли.

Переключение между двумя каталогами с помощью «cd -»

Предположим, вы р бот ли в отд ленном вложенном к т логе и з пустили `cd`, чтобы перейти куд -то еще:

```
$ pwd
/home/smith/Finances/Bank/Checking/Statements
$ cd /etc
```

но з тем з хотели вернуться в к т лог *Statements,* где только что были. Не вводите повторно длинный путь к к т логу. Просто з пустите `cd` с дефисом в к честве ргумент :

```
$ cd -
/home/smith/Finances/Bank/Checking/Statements
```

Эт ком нд возвр щ ет в ш экземпляр оболочки в предыдущий к т лог и выводит бсолютный путь, чтобы вы зн ли, где н ходитесь.

Чтобы перемещ ться туд -обр тно между п рой к т логов, несколько р з з пустите `cd`. Это экономит время, когд вы сосредоточены н р боте в двух к т лог х в одном экземпляре оболочки. Одн ко есть одн з гвоздк : оболочк з помин ет только один предыдущий к т лог з р з. Н пример, если вы переключ етесь между */usr/local/bin* и */etc*:

```
$ pwd
/usr/local/bin
$ cd /etc                  Экземпляр оболочки помнит /usr/local/bin
$ cd -                     Экземпляр оболочки помнит /etc
/usr/local/bin
$ cd -                     Экземпляр оболочки помнит /usr/local/bin
/etc
```

и вы з пуск ете `cd` без ргументов, чтобы перейти в в ш дом шний к т лог:

```
$ cd                       Экземпляр оболочки помнит /etc
```

оболочк теперь з был */usr/local/bin* к к предыдущий к т лог:

```
$ cd -                     Экземпляр оболочки помнит ваш домашний каталог
/etc
$ cd -                     Экземпляр оболочки помнит /etc
/home/smith
```

Следующий метод позволяет обойти это огр ничение.

Переключение между несколькими каталогами с помощью pushd и popd

Ком нд `cd` позволяет переключ ться между двумя к т лог ми, но что, если в м нужно отслежив ть три или более к т логов? Предположим, вы созд ете лок льный веб-с йт н своем компьютере с Linux. Эт з д ч ч сто подр зумев ет использов ние четырех или более к т логов:

- Р сположение ктивных, р звернутых веб-стр ниц, т ких к к */var/www/html*.
- К т лог конфигур ции веб-сервер , н пример */etc/apache2*.
- Р сположение SSL-сертифик тов, н пример */etc/ssl/certs*.
- В ш р бочий к т лог, н пример *~/Work/Projects/Web/src*.

Поверьте мне, утомительно постоянно вводить:

```
$ cd ~/Work/Projects/Web/src
$ cd /var/www/html
$ cd /etc/apache2
$ cd ~/Work/Projects/Web/src
$ cd /etc/ssl/certs
```

Если у в с дисплей с большой ди гон лью, вы можете упростить з д чу, открыв отдельное окно оболочки для к ждого к т лог . Но если вы р бот ете в одном экземпляре оболочки (ск жем, через соединение SSH), воспользуйтесь функцией оболочки, н зыв емой *стеком к т логов*. Он позволяет легко перемещ ться между несколькими к т лог ми, используя встроенные ком нды оболочки `pushd`, `popd` и `dirs`. Н изучение этой функции вы потр тите, может быть, 15 минут, пользу от увеличения скорости р боты будете получ ть всю жизнь[1].

Стек к т логов — это список к т логов, которые вы посетили в текущем экземпляре оболочки и решили отслежив ть. Вы упр вляете стеком, выполняя две опер ции, н зыв емые *вт лкив нием (pushing)* и *извлечением (popping)*. Вт лкив ние к т лог доб вляет его в н ч ло списк , который н зыв ется *вершиной* стек . Извлечение уд ляет верхний к т лог из стек [2]. Первон ч льно стек содержит только в ш текущий к т лог, но вы можете доб влять (вт лкив ть) и уд лять (извлек ть) к т логи и быстро переходить между ними.

К ждый экземпляр оболочки поддержив ет свой собственный стек к т логов.

[1] Альтерн тивой является открытие нескольких вирту льных дисплеев с помощью термин льных прогр мм, т ких к к `screen` и `tmux`, которые н зыв ются термин льными мультиплексор ми. Их изучение требует больше усилий, чем изучение стек к т логов, но н них тоже стоит обр тить вним ние.

[2] Если вы зн комы со стек ми из информ тики, стек к т логов — это т кой же стек, только для имен к т логов.

Я н чну с основных опер ций (вт лкив ние, извлечение, просмотр), з тем перейду к другим вещ м.

Поместить каталог в стек

Ком нд `pushd` (сокр щение от *push directory*) выполняет следующие действия:

1. Доб вляет з д нный к т лог в вершину стек .
2. Выполняет `cd` в этот к т лог.
3. Выводит н экр н стек сверху вниз для спр вки.

Я созд м стек из четырех к т логов, доб вляя их по одному:

```
$ pwd
/home/smith/Work/Projects/Web/src
$ pushd /var/www/html
/var/www/html ~/Work/Projects/Web/src
$ pushd /etc/apache2
/etc/apache2 /var/www/html ~/Work/Projects/Web/src
$ pushd /etc/ssl/certs
/etc/ssl/certs /etc/apache2 /var/www/html ~/Work/Projects/Web/src
$ pwd
/etc/ssl/certs
```

Оболочк печ т ет стек после к ждой ком нды `pushd`. Текущий к т лог — левый (верхний).

Просмотр стека каталогов

Выведите н экр н стек к т логов текущего экземпляр оболочки с помощью ком нды `dirs`. Он не изменяет стек:

```
$ dirs
/etc/ssl/certs /etc/apache2 /var/www/html ~/Work/Projects/Web/src
```

Если вы предпочит ете печ т ть стек сверху вниз, используйте п р метр `-p`:

```
$ dirs -p
/etc/ssl/certs
/etc/apache2
/var/www/html
~/Work/Projects/Web/src
```

Вы можете перед ть вывод ком нде `nl` для нумер ции строк с нуля:

```
$ dirs -p | nl -v0
     0 /etc/ssl/certs
     1 /etc/apache2
     2 /var/www/html
     3 ~/Work/Projects/Web/src
```

Это можно сдел ть еще проще, з пустив `dirs -v`. Этот п р метр выводит н экр н стек с пронумеров нными строк ми:

```
$ dirs -v
 0 /etc/ssl/certs
 1 /etc/apache2
 2 /var/www/html
 3 ~/Work/Projects/Web/src
```

Если вы предпочит ете т кой форм т, подум йте о созд нии псевдоним :

```
# Поместите в файл конфигурации оболочки и примените его
alias dirs='dirs -v'
```

Извлечь каталог из стека

Ком нд `popd` (*pop directory*) противоположн ком нде `pushd`. Он дел ет следующее:

1. Уд ляет один к т лог из вершины стек .
2. Выполняет ком нду `cd` в новый верхний к т лог.
3. Выводит н экр н стек сверху вниз для спр вки.

Н пример, если в в шем стеке четыре к т лог :

```
$ dirs
/etc/ssl/certs /etc/apache2 /var/www/html ~/Work/Projects/Web/src
```

Несколько р з з пустим ком нду `popd`, котор я будет проходить по этим к т лог м сверху вниз:

```
$ popd
/etc/apache2 /var/www/html ~/Work/Projects/Web/src
$ popd
/var/www/html ~/Work/Projects/Web/src
$ popd
~/Work/Projects/Web/src
$ popd
bash: popd: directory stack empty
$ pwd
~/Work/Projects/Web/src
```

Ком нды `pushd` и `popd` н столько экономят время, что я рекомендую созд в ть двухбуквенные псевдонимы, которые н бир ются т к же быстро, к к `cd`:

```
# Поместите в файл конфигурации оболочки и примените его
alias gd=pushd
alias pd=popd
```

Поменять местами каталоги в стеке

Теперь, когд вы умеете созд в ть и очищ ть стек к т логов, д в йте сосредоточимся н пр ктических пример х его использов ния. pushd без ргументов меняет мест ми дв верхних к т лог в стеке и переходит к новому верхнему к т логу. Д в йте несколько р з переместимся между */etc/apache2* и в шим р бочим к т логом, з пустив pushd. Посмотрите, к к третий к т лог */var/www/html* ост ется т м же, когд первые дв меняются мест ми:

```
$ dirs
/etc/apache2 ~/Work/Projects/Web/src /var/www/html
$ pushd
~/Work/Projects/Web/src /etc/apache2 /var/www/html
$ pushd
/etc/apache2 ~/Work/Projects/Web/src /var/www/html
$ pushd
~/Work/Projects/Web/src /etc/apache2 /var/www/html
```

Обр тите вним ние, что pushd при переключении между двумя к т лог ми н логичен ком нде cd, но не огр ничив ется з помин нием только одного к т лог .

Превратить ошибочный cd в pushd

Предположим, вы перемещ етесь между несколькими к т лог ми с помощью pushd, потом случ йно з пуск ете cd и теряете к т лог:

```
$ dirs
~/Work/Projects/Web/src /var/www/html /etc/apache2
$ cd /etc/ssl/certs
$ dirs
/etc/ssl/certs /var/www/html /etc/apache2
```

Ком нд cd з менил к т лог *~/Work/Projects/Web/src* в стеке н */etc/ssl/certs*. Но не волнуйтесь. Вы можете доб вить отсутствующий к т лог обр тно в стек, не вводя его длинный путь. Просто з пустите pushd дв жды, один р з с дефисом и один р з без:

```
$ pushd -
~/Work/Projects/Web/src /etc/ssl/certs /var/www/html /etc/apache2
$ pushd
/etc/ssl/certs ~/Work/Projects/Web/src /var/www/html /etc/apache2
```

Д в йте р зберем, почему это р бот ет:

- Перв я ком нд pushd возвр щ ет в с в предыдущий к т лог, посещенный в этом экземпляре оболочки *~/Work/Projects/Web/src*, и помещ ет его в стек. pushd, к к и cd, приним ет дефис в к честве ргумент , озн ч ющего «вернуться в мой предыдущий к т лог».

- Втор я ком нд pushd меняет мест ми дв верхних к т лог , возвр щ я в с в */etc/ssl/certs*. В результ те вы восст новили *~/Work/Projects/Web/src* н второй позиции в стеке, где он был бы, если бы вы не допустили ошибки.

Ком нд «ой, я з был pushd» дост точно полезн , чтобы д ть ей псевдоним. Я н зыв ю ее slurp (потому что он «slurps back[1]» потерянный мной к т лог):

```
# Поместите в файл конфигурации оболочки и примените его
alias slurp='pushd - && pushd'
```

Углубиться в стек

Что дел ть, если вы хотите перейти в другие к т логи в стеке, помимо двух верхних? pushd и popd приним ют положительный или отриц тельный целочисленный ргумент для д льнейшей р боты со стеком. Ком нд

```
$ pushd +N
```

сдвиг ет N к т логов с вершины стек вниз, з тем переходит к новому верхнему к т логу. Отриц тельный ргумент (-N) сдвиг ет к т логи в противоположном н пр влении, снизу вверх, перед выполнением cd:

```
$ dirs
/etc/ssl/certs ~/Work/Projects/Web/src /var/www/html /etc/apache2
$ pushd +1
~/Work/Projects/Web/src /var/www/html /etc/apache2 /etc/ssl/certs
$ pushd +2
/etc/apache2 /etc/ssl/certs ~/Work/Projects/Web/src /var/www/html
```

Т ким обр зом, вы можете перейти к любому другому к т логу в стеке с помощью простой ком нды. Одн ко если в ш стек длинный, трудно определить числовую позицию нужного к т лог . Поэтому выведите числовую позицию к ждого к т лог с помощью dirs -v, к к вы дел ли в р зделе «Просмотр стек к т логов» н с. 82:

```
$ dirs -v
 0 /etc/apache2
 1 /etc/ssl/certs
 2 ~/Work/Projects/Web/src
 3 /var/www/html
```

Чтобы переместить */var/www/html* н вершину стек (и сдел ть его текущим к т логом), з пустите pushd +3.

[1] «Выплевыв ет». — *Примеч. пер.*

Чтобы перейти к к т логу в нижней ч сти стек , з пустите `pushd -0`:

```
$ dirs
/etc/apache2 /etc/ssl/certs ~/Work/Projects/Web/src /var/www/html
$ pushd -0
/var/www/html /etc/apache2 /etc/ssl/certs ~/Work/Projects/Web/src
```

Вы т кже можете уд лить к т логи из стек , используя `popd` с числовым ргументом. Ком нд

```
$ popd +N
```

уд ляет к т лог, р сположенный н позиции `N`, из стек , счит я сверху вниз. Отриц тельный ргумент (`-N`) озн ч ет отсчет снизу стек . Подсчет н чин ется с нуля, поэтому `popd +1` уд ляет второй к т лог сверху:

```
$ dirs
/var/www/html /etc/apache2 /etc/ssl/certs ~/Work/Projects/Web/src
$ popd +1
/var/www/html /etc/ssl/certs ~/Work/Projects/Web/src
$ popd +2
/var/www/html /etc/ssl/certs
```

Резюме

Все приемы, опис нные в этой гл ве, легко освоить, если немного попр ктиков ться, и они сэкономят в м много времени и усилий. Вот техники, которые будут особенно полезны:

- `CDPATH` для быстрой н виг ции.
- `pushd` и `popd` для быстрого возвр т .
- Испр вление случ йно з пущенной ком нды `cd`.

ЧАСТЬ 2

Продвинутые навыки

Теперь, когд вы поним ете основы ком нд, конвейеров, оболочки и н виг ции по ф йловой системе, пришло время сдел ть ш г вперед. В следующих пяти гл в х я позн комлю в с с множеством новых прогр мм для Linux и некоторыми в жными концепциями оболочки. Вы н учитесь применять их для созд ния сложных ком нд и решения ре льных з д ч н компьютере с Linux.

ГЛАВА 5

Расширяем ваш инструментарий

Системы, основ нные н Linux, пост вляются с тысяч ми встроенных в ком ндную строку ком нд. Опытные пользов тели обычно пол г ются н меньший н бор — основной инструмент рий, к которому они возвр щ ются снов и снов . Гл в 1 доб вил в в ш н бор шесть весьм полезных ком нд, теперь я позн комлю в с с еще примерно дюжиной. Я кр тко опишу к ждую ком нду и пок жу несколько примеров ее использов ния (чтобы увидеть все доступные п р метры, просмотрите спр вочную стр ницу соответствующей ком нды). Я т кже позн комлю в с с двумя мощными ком нд ми, которые труднее освоить, но они того стоят: `awk` и `sed`. В целом, ком нды в этой гл ве служ т четырем общим пр ктическим з д ч м конвейеров и других сложных ком нд.

Созд ние текст

Вывод д т, времени, последов тельностей цифр и букв, путей к ф йл м, повторяющихся строк и другого текст для быстрого з пуск конвейеров.

Извлечение текст

Извлечение к кой-либо ч сти текстового ф йл с помощью комбин ции `grep`, `cut`, `head` или `tail` с функцией `awk`.

Объединение текст

Объединение текст из р зных ф йлов с помощью `cat` и `tac` или `echo` и `paste`. Вы т кже можете чередов ть текстовые ф йлы с помощью `paste` и `diff`.

Преобр зов ние текст

Преобр зов ние текст с помощью простых ком нд, т ких к к `tr` и `rev`, или более мощных ком нд, т ких к к `awk` и `sed`.

Эт гл в предст вляет собой кр ткий обзор. В последующих гл в х вы позн комитесь с пр ктическим использов нием этих ком нд.

Создание текста

К ждый конвейер н чин ется с простой ком нды, котор я выводит д нные в ст нд ртный вывод. Иногд это ком нд вроде `grep` или `cut`, котор я извлек ет определенные д нные из ф йл :

```
$ cut -d: -f1 /etc/passwd | sort          Вывести все имена пользователей
                                          и отсортировать их
```

Или ком нд `cat` для перед чи полного содержимого нескольких ф йлов другим ком нд м:

```
$ cat *.txt | wc -l                       Общее количество строк
```

Исходный текст может поступ ть в конвейер и из других источников. Вы уже зн ете ком нду `ls`, котор я печ т ет имен ф йлов и к т логов и связ нную с ними информ цию. Д в йте взглянем н некоторые другие ком нды и методы созд ния текст :

`date`
: Вывод д ты и времени в р зличных форм т х.

`seq`
: Вывод последов тельности чисел.

Р сширение ком нд с помощью фигурных скобок
: Функция оболочки, котор я выводит последов тельность цифр или символов.

`find`
: Выводит путь к ф йлу.

`yes`
: Повторно печ т ет одну и ту же строку.

Команда date

Ком нд `date` выводит текущую д ту и/или время в р зличных форм т х:

```
$ date                                    Формат по умолчанию
Mon Jun 28 16:57:33 EDT 2021
$ date +%Y-%m-%d                          Формат год-месяц-день
2021-06-28
```

```
$ date +%H:%M:%S                    Формат часы:минуты:секунды
16:57:33
```

Чтобы упр влять форм том вывод , ук жите ргумент, н чин ющийся со зн к плюс, з которым следует любой текст. Текст может содерж ть специ льные выр жения, н чин ющиеся со зн к процент , н пример `%Y` для четырехзн чного зн чения текущего год и `%H` для текущего ч с в 24-ч совом форм те. Полный список выр жений н ходится н спр вочной стр нице ком нды `date`.

```
$ date +»I cannot believe it's already %A!»      День недели
I cannot believe it's already Tuesday!
```

Команда seq

Ком нд `seq` печ т ет последов тельность чисел из ди п зон . Ук жите дв ргумент , нижнее и верхнее зн чения ди п зон , и `seq` н печ т ет весь ди п зон:

```
$ seq 1 5                           Выводит все целые числа от 1 до 5 включительно
1
2
3
4
5
```

Если вы з д ете три ргумент , первый и третий определяют ди п зон, среднее число — это ш г:

```
$ seq 1 2 10                        Увеличение на 2 вместо 1
1
3
5
7
9
```

Используйте отриц тельный ш г, н пример `-1`, для созд ния уменьш ющейся последов тельности:

```
$ seq 3 -1 0
3
2
1
0
```

или дробный ш г для получения чисел с пл в ющей з пятой:

```
$ seq 1.1 0.1 2                    Увеличение на 0,1
1.1
1.2
1.3
⋮
2.0
```

По умолч нию зн чения р зделяются символом новой строки, но вы можете изменить р зделитель с помощью п р метр `-s`, после которого можно ук з ть любой символ н в ш выбор:

```
$ seq -s/ 1 5                      Разделение значений с помощью косой черты
1/2/3/4/5
```

П р метр `-w` приводит все зн чения к один ковой ширине (в символ х), доб вляя ведущие нули по мере необходимости:

```
$ seq -w 8 10
08
09
10
```

`seq` может выводить числ во многих других форм т х (см. спр вочную стр - ницу), но мои примеры пок зыв ют н иболее р спростр ненные в ри нты использов ния.

Расширение команд с помощью фигурных скобок

Ком ндн я оболочк предост вляет собственный способ вывод последов - тельности чисел, известный к к *р сширение фигурных скобок* (*brace expansion*). Н чните с левой фигурной скобки, доб вьте дв целых числ , р зделенных двумя точк ми, и з кончите пр вой фигурной скобкой:

```
$ echo {1..10}                     Вперед, начиная с 1
1 2 3 4 5 6 7 8 9 10
$ echo {10..1}                     Назад, начиная с 10
10 9 8 7 6 5 4 3 2 1
$ echo {01..10}                    С ведущими нулями (для равной ширины)
01 02 03 04 05 06 07 08 09 10
```

В более общем случ е выр жение оболочки `{x..y..z}` генерирует зн чения от *x* до *y* с ш гом *z*:

```
$ echo {1..1000..100}                    Приращение сотнями, начиная с 1
1 101 201 301 401 501 601 701 801 901
$ echo {1000..1..100}                    Уменьшение сотнями, начиная с 1000
1000 900 800 700 600 500 400 300 200 100
$ echo {01..1000..100}                   С ведущими нулями
0001 0101 0201 0301 0401 0501 0601 0701 0801 0901
```

Фигурные скобки vs квадратные

Кв др тные скобки — это опер тор сопост вления имен ф йлов с ш - блоном (см. гл ву 2). Р сширение фигурных скобок ник к не связ но с имен ми ф йлов. Это просто вычисление последов тельности строковых зн чений. Вы можете использов ть р сширение фигурных скобок для *вывод* имен ф йлов, но сопост вления с ш блоном при этом не происходит:

```
$ ls
file1 file2 file4
$ ls file[2-4]                           Соответствует существующим именам
file2 file4                              файлов
$ ls file{2..4}                          Вычисляется в: file2 file3 file4
ls: cannot access 'file3': No such file or directory
file2 file4
```

Р сширение с помощью фигурных скобок т кже может созд в ть последов тельности букв, которые не может вывести ком нд `seq`:

```
$ echo {A..Z}
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
```

Р сширение с помощью фигурных скобок всегд з писыв ет вывод в виде одной строки, р зделенной пробел ми. Изменить это можно, н пр вив вывод другим ком нд м, т ким к к `tr` (см. подр здел «Ком нд `tr`» н с. 107):

```
$ echo {A..Z} | tr -d ' '                Удалить пробелы
ABCDEFGHIJKLMNOPQRSTUVWXYZ
$ echo {A..Z} | tr ' ' '\n'              Заменить пробелы на символ новой
A                                        строки
B
C
⋮
Z
```

Созд йте псевдоним, который печ т ет *n-ю* букву нглийского лф вит :

```
$ alias nth="echo {A..Z} | tr -d ' ' | cut -c"
$ nth 10
J
```

Команда find

Ком нд `find` рекурсивно выводит список ф йлов в к т логе, спуск ясь по подк т лог м и выводя полные пути[1]. Результ ты выводятся не в лф витном порядке (при необходимости отпр вьте вывод в ком нду `sort`):

```
$ find /etc -print                                  Список всех каталогов в /etc
/etc                                                рекурсивно
/etc/issue.net
/etc/nanorc
/etc/apache2
/etc/apache2/sites-available
/etc/apache2/sites-available/default.conf
⋮
```

`find` имеет множество опций, которые вы можете комбиниров ть. Р ссмотрим несколько н иболее полезных.

Огр ничьте вывод только ф йл ми или к т лог ми с опцией `-type`:

```
$ find . -type f -print                             Только файлы
$ find . -type d -print                             Только каталоги
```

Огр ничьте вывод имен ми, которые соответствуют ш блону имени ф йл с опцией `-name`. З ключите ш блон в к вычки или экр нируйте, чтобы оболочк его не вычислял :

```
$ find /etc -type f -name "*.conf" -print           Файлы, заканчивающиеся на .conf
/etc/logrotate.conf
/etc/systemd/logind.conf
/etc/systemd/timesyncd.conf
⋮
```

Сдел йте соответствие имен ш блону нечувствительным к регистру букв с помощью опции `-iname`:

```
$ find . -iname "*.txt" -print
```

`find` т кже может выполнить ком нду Linux для всех ф йлов в выходных д нных, используя `-exec`. Синт ксис выглядит несколько громоздким:

1. Н берите ком нду `find` без п р метр `-print`.
2. Доб вьте п р метр `-exec`, з тем ком нду для выполнения. Используйте скобки `{}`, чтобы ук з ть, где в ком нде должен отобр ж ться путь к ф йлу.
3. В конце выр жения используйте точку с з пятой в к вычк х `";"` или экр ниров нную `\;`.

[1] Родственн я ком нд `ls -R` выд ет д нные в форм те, менее удобном для использов ния в конвейер х.

Вот простой пример вывод символ @ по обеим сторон м пути к ф йлу:

```
$ find /etc -exec echo @ {} @ ";"
@ /etc @
@ /etc/issue.net @
@ /etc/nanorc @
⋮
```

В более пок з тельном примере все ф йлы с р сширением *.conf* в к т логе */etc* и его подк т лог х выводятся в форм те длинного списк (`ls -l`):

```
$ find /etc -type f -name "*.conf" -exec ls -l {} ";"
-rw-r--r-- 1 root root 703     Aug 21 2017 /etc/logrotate.conf
-rw-r--r-- 1 root root 1022    Apr 20 2018 /etc/systemd/logind.conf
-rw-r--r-- 1 root root 604     Apr 20 2018 /etc/systemd/timesyncd.conf
⋮
```

`find -exec` можно использов ть для уд ления ф йлов во вложенных к т лог х (но будьте осторожны!). Д в йте уд лим ф йлы с имен ми, ок нчив ющимися н тильду (~), в к т логе *$HOME/tmp* и его подк т лог х. В целях безоп сности сн ч л з пустите ком нду `echo rm`, чтобы увидеть, к кие ф йлы будут уд лены, з тем уберите `echo` из выр жения, чтобы уд лить по-н стоящему:

```
$ find $HOME/tmp -type f -name "*~" -exec echo rm {} ";"  Добавим echo
rm /home/smith/tmp/file1~
rm /home/smith/tmp/junk/file2~
rm /home/smith/tmp/vm/vm-8.2.0b/lisp/vm-cus-load.el~
$ find $HOME/tmp -type f -name "*~" -exec rm {} ";"       Удалить по-настоящему
```

Команда yes

Ком нд `yes` выводит одну и ту же строку снов и снов , пок вы ее не ост новите:

```
$ yes                                    Выводит «y» по умолчанию
y
y
y ^C                                     Остановить команду с помощью Ctrl-C
$ yes woof!                              Повторять любую другую строку
woof!
woof!
woof! ^C
```

К к я пр ктическ я польз от этого? yes может вводить д нные для интер ктивных прогр мм, чтобы они могли р бот ть без в шего вмеш тельств . Н пример, прогр мм fsck, котор я проверяет ф йловую систему Linux н н личие ошибок, может спросить пользов теля о необходимости продолжения и будет ожид ть ответ y или n. Выходные д нные ком нды yes, перед нные в fsck, отвеч ют н к ждое пригл шение от в шего имени, т к что вы можете уйти и позволить fsck выполниться до конц [1].

Основное использов ние ком нды yes в н ших пример х — печ ть строки определенное количество р з путем перед чи yes в head (вы увидите пр ктический пример использов ния в р зделе «Созд ние тестовых ф йлов» н с. 177):

```
$ yes "Efficient Linux" | head -n3          Вывести строку 3 раза
Efficient Linux
Efficient Linux
Efficient Linux
```

Извлечение текста

Когд в м нужн только ч сть ф йл , с мые простые ком нды для объединения в конвейер — это grep, cut, head и tail. Вы уже видели первые три в действии в гл ве 1: grep выводит строки, соответствующие з д нной строке; cut выводит столбцы из ф йл ; head выводит первые строки ф йл . Нов я ком нд tail является противоположностью head и печ т ет последние строки ф йл . Н рис. 5.1 пок з н совместн я р бот этих четырех ком нд.

В этой гл ве р ссмотрим подробно ком нду grep, котор я может дел ть зн чительно больше, чем просто сопост вление строк. Более форм льно будет объяснен tail. Т кже ост новимся н функции ком нды awk для извлечения столбцов способом, недоступным для cut. Комбин ция этих пяти ком нд может извлечь пр ктически любой текст с помощью одного конвейер .

[1] Некоторые современные ре лиз ции fsck имеют опции -y и -n для ответов «д » или «нет» н к ждое пригл шение, поэтому ком нд yes для них не требуется.

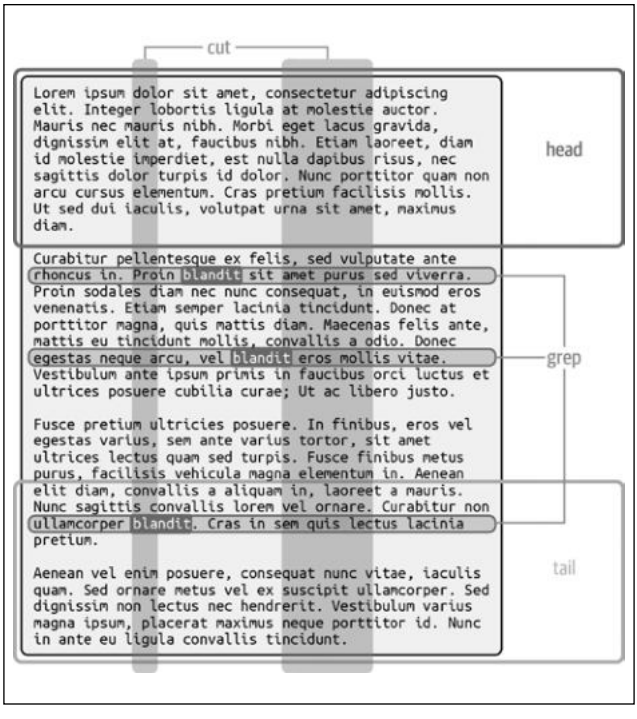


Рис. 5.1. `head`, `grep` и `tail` извлекают строки и столбцы. В этом примере `grep` соответствует строкам, содержащим слово *blandit*

Команда grep. Более глубокий взгляд

Вы уже видели, к к `grep` выводит из ф йл строки, соответствующие з д нному ш блону:

```
$ cat frost
Whose woods these are I think I know.
His house is in the village though;
He will not see me stopping here
To watch his woods fill up with snow.
This is not the end of the poem.
$ grep his frost                          Вывести строки, содержащие «his»
To watch his woods fill up with snow.
This is not the end of the poem.          «This» содержит «his»
```

`grep` имеет несколько очень полезных п р метров. Используйте `-w` для н хождения соответствий только целым слов м:

```
$ grep -w his frost                          Искать точное соответствие «his»
To watch his woods fill up with snow.
```

Используйте п р метр `-i`, чтобы игнориров ть регистр букв:

```
$ grep -i his frost
His house is in the village though;          Содержит «His»
To watch his woods fill up with snow.        Содержит «his»
This is not the end of the poem.             «This» содержит «his»
```

Используйте п р метр `-l`, чтобы вывести только имен ф йлов, содерж щих совп д ющие строки, но не с ми строки:

```
$ grep -l his *                              В каком файле содержится «his»?
frost
```

Одн ко н стоящ я сил `grep` проявляется, когд вы переходите от сопост вления простых строк к сопост влению ш блонов, н зыв емых *регулярными выр жениями*[1]. В этом случ е синт ксис отлич ется от ш блонов имен ф йлов. Ч стичное опис ние синт ксис приведено в т бл. 5.1.

Таблица 5.1. Синтаксис регулярных выражений, используемый командами `grep`, `awk` и `sed`[2]

Соответствие	Используемые выражения	Пример
Начало строки	`^`	`^a` = строка, начинающаяся с *a*
Конец строки	`$`	`!$` = строка, заканчивающаяся восклицательным знаком
Любой одиночный символ (кроме новой строки)	`.`	`...` = любые три последовательных символа
Знаки вставки, доллара или любой другой специальный символ *c*	`\c`	`$` = знак доллара
Ноль или более вхождений выражения *E*	`E*`	`_*` = ноль или более знаков подчеркивания
Любой одиночный символ в наборе	`[characters]`	`[aeiouAEIOU]` = любая гласная

[1] Н зв ние grep является сокр щением от «получить регулярное выр жение и р спеч т ть» (get regular expression and print).

[2] Эти три ком нды р злич ются в обр ботке регулярных выр жений. В т бл. 5.1 предст влен неполный список выр жений.

Соответствие	Используемые выражения	Пример
Любой одиночный символ, не входящий в набор	`[^characters]`	`[^aeiouAEIOU]` = любая не гласная
Любой символ в диапазоне между *c1* и *c2*	`[c1-c2]`	`[0-9]` = любая цифра
Любой символ вне диапазона между *c1* и *c2*	`[^c1-c2]`	`[^0-9]` = любой нецифровой символ
Любое из двух выражений *E1* или *E2*	`E1|E2` для `grep` и `sed`, `E1/E2` для `awk`	`one|two` = или *one*, или *two* `one|two` = или *one*, или *two*
Группировка выражения *E* с учетом приоритета	`\(E\)` для `grep` и `sed`[1], `(E)` для `awk`	`\(one|two\)\*` = ноль или более вхождений *one* или *two*, `(one|two)*` = ноль или более вхождений *one* или *two*

Рассмотрим несколько примеров команд `grep` с регулярными выражениями.

Найти совпадение всех строк, начинающихся с заглавной буквы:

```
$ grep '^[A-Z]' myfile
```

Найти совпадение со всеми непустыми строками (то есть совпадение с пустыми строками и их пропуск благодаря параметру `-v`):

```
$ grep -v '^$' myfile
```

Найти все строки, содержащие либо *cookie*, либо *cake*:

```
$ grep 'cookie\|cake' myfile
```

Найти все строки длиной не менее пяти символов:

```
$ grep '.....' myfile
```

Найти все строки, в которых символ «меньше» появляется перед символом «больше», например строки кода HTML:

```
$ grep '<.*>' page.html
```

[1] Для `sed` этот синтаксис делает больше, чем просто группировка, см. «Сопоставление подвыражений sed» на с. 116.

Регулярные выр жения прекр сны, но иногд они приводят к неожид нным результ т м. Предположим, вы хотите н йти в ф йле *frost* две строки, содерж щие букву `w`, з которой следует точк . Следующ я ком нд д ет непр вильные результ ты, поскольку точк — это регулярное выр жение, озн ч ющее «любой символ»:

```
$ grep w. frost
Whose woods these are I think I know.
He will not see me stopping here
To watch his woods fill up with snow.
```

Чтобы обойти эту проблему, вы можете экр ниров ть специ льный символ:

```
$ grep 'w\.' frost
Whose woods these are I think I know.
To watch his woods fill up with snow.
```

Т кое решение ст новится громоздким, если в м нужно экр ниров ть много специ льных символов. К сч стью, вы можете з ст вить `grep` з быть о регулярных выр жениях и иск ть букв льно к ждый символ во входных д нных, используя п р метр `-F` (fixed). Или, в к честве льтерн тивы с т ким же результ том, з пустите `fgrep` вместо `grep`:

```
$ grep -F w. frost
Whose woods these are I think I know.
To watch his woods fill up with snow.
$ fgrep w. frost
Whose woods these are I think I know.
To watch his woods fill up with snow.
```

`grep` имеет много других п р метров. Р спростр ненную проблему реш ет п р метр `-f` (строчный символ, не пут йте его с `F`), служ щий для сопост вления с н бором строк, не с одной строкой. В к честве пр ктического пример д в йте перечислим все оболочки, н йденные в ф йле */etc/passwd*, который я предст вил в р зделе «Ком нд #5: `sort`» н с. 28. К к вы помните, к жд я строк в */etc/passwd* содержит информ цию о пользов теле, двоеточия служ т р зделителями полей. Последнее поле в к ждой строке — имя прогр ммы, з пуск емой при входе пользов теля в систему. Эт прогр мм з ч стую, но не всегд , является оболочкой:

```
$ cat /etc/passwd
root:x:0:0:root:/root: /bin/bash                    Седьмое поле — оболочка
daemon:x:1:1:daemon:/usr/sbin: /usr/sbin/nologin    Седьмое поле — не оболочка
⋮
```

К к узн ть, является ли прогр мм оболочкой? В ф йле */etc/shells* перечислены все допустимые оболочки вход в систему Linux:

```
$ cat /etc/shells
/bin/sh
/bin/bash
/bin/csh
```

Т ким обр зом, вы можете перечислить все действующие оболочки из */etc/passwd*, извлек я седьмое поле с помощью `cut`, уд ляя дублик ты с помощью `sort -u` и сверяя результ ты с */etc/shells* с помощью `grep -f`. Т кже доб вим п р метр `-F`, т к к к все строки в */etc/shells* восприним ются букв льно, д же если они содерж т специ льные символы:

```
$ cut -d: -f7 /etc/passwd | sort -u | grep -f /etc/shells -F
/bin/bash
/bin/sh
```

Команда tail

Ком нд `tail` печ т ет последние строки ф йл — по умолч нию 10 строк. Это сестр ком нды `head`. Предположим, у в с есть ф йл с именем *alphabet*, содерж щий 26 строк, по одной н букву:

```
$ cat alphabet
A is for aardvark
B is for bunny
C is for chipmunk
⋮
X is for xenorhabdus
Y is for yak
Z is for zebu
```

Выведите последние три строки с помощью `tail`. Опция `-n` з д ет количество печ т емых строк, к к и для ком нды `head`:

```
$ tail -n3 alphabet
X is for xenorhabdus
Y is for yak
Z is for zebu
```

Если пост вить перед номером зн к `+`, печ ть н чнется со строки этого номер и продолжится до конц ф йл . Следующ я ком нд н чин ется с 25-й строки ф йл :

```
$ tail -n+25 alphabet
Y is for yak
Z is for zebu
```

Объедините `tail` и `head`, чтобы напечатать любой диапазон строк из файла. Например, чтобы напечатать только четвертую строку, извлеките первые четыре строки и выведите последнюю:

```
$ head -n4 alphabet | tail -n1
D is for dingo
```

В общем, чтобы напечатать строки от `M` до `N`, извлеките первые `N` строк с помощью `head`, затем выведите последние `N-M+1` строк с помощью `tail`. Выведите строки с шестой по восьмую файла *alphabet*:

```
$ head -n8 alphabet | tail -n3
F is for falcon
G is for gorilla
H is for hawk
```

И `head`, и `tail` поддерживают более простой синтаксис для указания количества строк без параметра `-n`. Этот синтаксис — недокументированный и устаревший, но, вероятно, будет поддерживаться всегда:

```
$ head -4 alphabet        То же самое, что и head -n4 alphabet
$ tail -3 alphabet        То же самое, что и tail -n3 alphabet
$ tail +25 alphabet       То же самое, что и tail -n+25 alphabet
```

Команда awk {print}

Командa `awk` представляет собой обработчик текста общего назначения с сотнями применений. Давайте рассмотрим одну небольшую функцию, `print`, которая извлекает столбцы из файла способами, недоступными для `cut`. Рассмотрим системный файл */etc/hosts*, который включает IP-адреса и имена хостов, разделенные любым количеством пробелов:

```
$ less /etc/hosts
127.0.0.1     localhost
127.0.1.1   myhost        myhost.example.com
192.168.1.2  frodo
192.168.1.3       gollum
192.168.1.28   gandalf
```

Предположим, вы хотите вывести на экран имена хостов — второе слово в каждой строке. Проблема в том, что каждому имени хоста предшествует произвольное количество пробелов. `cut` требует, чтобы столбцы были либо аккуратно выровнены (`-c`), либо разделены одним определенным символом (`-f`). В таком же нужна команда для печати второго слова в каждой строке, и ее предоставляет `awk`:

```
$ awk '{print $2}' /etc/hosts
localhost
myhost
frodo
gollum
gandalf
```

awk ссыл ется н любой столбец зн ком долл р , з которым следует номер столбц , н пример $7 для седьмого столбц . Если номер столбц состоит из нескольких цифр, з ключите число в круглые скобки, н пример $(25). Чтобы обр титься к последнему полю, используйте $NF (number of fields, то есть число полей). Чтобы обр титься ко всей строке, используйте $0.

awk по умолч нию не печ т ет пробелы между зн чениями. Если в м нужны пробелы, р зделите зн чения з пятыми:

```
$ echo Efficient fun Linux | awk '{print $1 $3}'          Без пробела
EfficientLinux
$ echo Efficient fun Linux | awk '{print $1, $3}'         С пробелом
Efficient Linux
```

Опер тор print ком нды awk отлично подходит для обр ботки вывод ком нд, д же если колонки совсем не ккур тны. Примером может служить ком нд df, котор я выводит объем свободного и з нятого дискового простр нств в системе Linux:

```
$ df / /data
Filesystem   1K-blocks    Used         Available    Use% Mounted on
/dev/sda1    1888543276   902295944    890244772    51% /
/dev/sda2    7441141620   1599844268   5466214400   23% /data
```

Р сположение столбцов может р злич ться в з висимости от длины путей в столбце Filesystems, р змеров диск и п р метров, которые вы перед ете df, поэтому вы не можете с уверенностью извлек ть зн чения с помощью cut. Одн ко с помощью awk можно легко вывести, н пример, четвертое зн чение в к ждой строке с д нными о доступном месте н диске:

```
$ df / /data | awk '{print $4}'
Available
890244772
5466214400
```

При этом можно не выводить первую строку (з головок):

```
$ df / /data | awk ' FNR>1 {print $4}'
890244772
5466214400
```

Если вы столкнулись с вводом, р зделенным чем-то другим, кроме пробелов, awk с п р метром -F может изменить р зделитель полей н любое регулярное выр жение:

```
$ echo efficient:::::linux | awk -F':*' '{print $2}'   Любое количество двоеточий
linux
```

Вы узн ете больше о ком нде awk в р зделе «Основы awk» н с. 109.

Объединение текста

Вы уже зн ете несколько ком нд, объединяющих текст из р зных ф йлов. Перв я — это cat, котор я выводит содержимое нескольких ф йлов. Это последов тельное объединение текстов «сверху вниз». Отсюд и его н зв ние — он объединяет (con*cat*enates) ф йлы:

```
$ cat poem1
It is an ancient Mariner,
And he stoppeth one of three.
$ cat poem2
'By thy long grey beard and glittering eye,
$ cat poem3
Now wherefore stopp'st thou me?
$ cat poem1 poem2 poem3
It is an ancient Mariner,
And he stoppeth one of three.
'By thy long grey beard and glittering eye,
Now wherefore stopp'st thou me?
```

Втор я ком нд для объединения текст , которую вы видели, — это встроенн я в оболочку echo. Он печ т ет любые перед нные в ми ргументы, р зделяя их одним пробелом, и соединяет тексты в одну строку:

```
$ echo efficient linux in $HOME
efficient linux in /home/smith
```

Д в йте р ссмотрим еще несколько ком нд, объединяющих текст:

tac — последов тельное объединение текстовых ф йлов.

paste — объединение строк текстовых ф йлов.

diff — ком нд , котор я чередует текст из двух ф йлов, отобр ж я только их р зличия.

Команда tac

Ком нд tac построчно перевор чив ет к ждый текст. Ее имя — это н пис нное з дом н перед н зв ние ком нды cat.

```
$ cat poem1 poem2 poem3 | tac
Now wherefore stopp'st thou me?
'By thy long grey beard and glittering eye,
And he stoppeth one of three.
It is an ancient Mariner,
```

Обр тите вним ние, что три ф йл были объединены, прежде чем перевернуть текст. Если вместо этого ук з ть ком нде tac несколько ф йлов в к честве р-гументов, он перевернет строки к ждого ф йл по очереди, выведя н экр н другие д нные:

```
$ tac poem1 poem2 poem3
And he stoppeth one of three.                    Первый файл перевернут
It is an ancient Mariner,
'By thy long grey beard and glittering eye,      Второй файл
Now wherefore stopp'st thou me?                  Третий файл
```

tac отлично подходит для обр ботки д нных, которые уже н ходятся в определенном порядке, но не могут быть отсортиров ны с помощью ком нды sort -r. Типичным случ ем является преобр зов ние ф йл журн л веб-сервер для обр ботки его строк от с мых новых к с мым ст рым:

```
192.168.1.34 - - [30/Nov/2021:23:37:39 -0500] "GET / HTTP/1.1" ...
192.168.1.10 - - [01/Dec/2021:00:02:11 -0500] "GET /notes.html HTTP/1.1" ...
192.168.1.8 - - [01/Dec/2021:00:04:30 -0500] "GET /stuff.html HTTP/1.1" ...
⋮
```

Строки р сположены в хронологическом порядке с отметк ми времени, но не в лф витном или числовом порядке, поэтому ком нд sort -r бесполезн . Ком нд tac может перевернуть эти строки, не обр щ я вним ния н метки времени.

Команда paste

Ком нд paste объединяет строки текстов в столбцы, р зделенные одним символом т буляции. Это сестр ком нды cut, котор я извлек ет р зделенные т буляцией столбцы из текст :

```
$ cat title-words1
EFFICIENT
AT
COMMAND
```

```
$ cat title-words2
linux
the
line
$ paste title-words1 title-words2
EFFICIENT    linux
AT  the
COMMAND  line
$ paste title-words1 title-words2 | cut -f2     cut и paste дополняют друг друга
linux
the
line
```

Измените разделитель на другой символ, например запятую, с помощью параметра `-d` (delimiter):

```
$ paste -d, title-words1 title-words2
EFFICIENT,linux
AT,the
COMMAND,line
```

Измените вывод, поменяв последовательность объединения с помощью параметра `-s`:

```
$ paste -d, -s title-words1 title-words2
EFFICIENT,AT,COMMAND
linux,the,line
```

Команда `paste` также может чередовать строки из двух или более файлов, если вы измените разделитель на символ новой строки (`\n`):

```
$ paste -d "\n" title-words1 title-words2
EFFICIENT
linux
AT
the
COMMAND
line
```

Команда diff

Команда `diff` сравнивает два файла построчно и выводит краткий отчет об их различиях:

```
$ cat file1
Linux is all about efficiency.
I hope you will enjoy this book.
$ cat file2
MacOS is all about efficiency.
```

```
I hope you will enjoy this book.
Have a nice day.
$ diff file1 file2
1c1
< Linux is all about efficiency.
---
> MacOS is all about efficiency.
2a3
> Have a nice day.
```

Код `1c1` означ ет, что строк 1 в первом ф йле отлич ется от строки 1 во втором ф йле. З этим кодом следует соответствующ я строк из *file1*, р зделитель из трех дефисов (`---`) и соответствующ я строк из *file2*. Н ч льный символ `<` всегд ук зыв ет н строку из первого ф йл , `>` ук зыв ет н строку из второго ф йл .

Код `2a3` означ ет, что в ф йле *file2* есть третья строк , отсутствующ я после второй строки ф йл *file1*. З этим обозн чением следует дополнительн я строк из ф йл 2: *Have a nice day*.

Вывод `diff` может содерж ть другие обозн чения и приним ть другие формы. Одн ко этого кр ткого объяснения дост точно для н шей основной цели — использов ния `diff` для чередов ния строк из двух ф йлов. Многие пользов тели не дум ли о т ком применении `diff`, но эт ком нд отлично подходит для формиров ния конвейеров при решении определенных з д ч. Н пример, вы можете вывести р зные строки с помощью `diff`, `grep` и `cut`:

```
$ diff file1 file2 | grep '^[<>]'
< Linux is all about efficiency.
> MacOS is all about efficiency.
> Have a nice day.
$ diff file1 file2 | grep '^[<>]' | cut -c3-
Linux is all about efficiency.
MacOS is all about efficiency.
Have a nice day.
```

Пр ктические примеры приведены в р здел х «Способ #4: Подст новк процесс » н с. 140 и «Проверк совп д ющих п р ф йлов» н с. 172.

Преобразование текста

В гл ве 1 было предст влено несколько ком нд, которые считыв ют текст со ст нд ртного ввод и преобр зуют его в ст нд ртном выходе. Ком нд `wc` выводит количество строк, слов и символов; `sort` упорядочив ет строки в л фв витном или числовом порядк х; `uniq` объединяет повторяющиеся строки. Д в йте обсудим еще несколько ком нд, которые преобр зуют ввод:

`tr` — преобр зует одни символы в другие.

`rev` — перевор чив ет символы в строке з дом н перед.

`awk` и `sed` — преобр зов тели текст общего н зн чения.

Команда tr

Ком нд `tr` переводит один н бор символов в другой. В гл ве 2 мы видели пример преобр зов ния двоеточий в символы новой строки для вывод переменной `PATH`:

```
$ echo $PATH | tr : "\n"                Преобразование двоеточий в символы новой
/home/smith/bin                         строки
/usr/local/bin
/usr/bin
/bin
/usr/games
/usr/lib/java/bin
```

`tr` приним ет дв н бор символов в к честве ргументов и переводит элементы первого н бор в соответствующие элементы второго. Ч сто выполняется преобр зов ние текст в верхний или нижний регистр:

```
$ echo efficient | tr a-z A-Z           Перевод a в A, b в B и т. д.
EFFICIENT
$ echo Efficient | tr A-Z a-z
efficient
```

Преобр зов ние пробелов в символы новой строки:

```
$ echo Efficient Linux | tr " " "\n"
Efficient
Linux
```

Уд ление пробелов с помощью п р метр `-d`:

```
$ echo efficient linux | tr -d ' \t'    Удаление пробелов и знаков табуляции
efficientlinux
```

Команда rev

Ком нд `rev` перевор чив ет символы з дом н перед в к ждой строке ввод [1]:

```
$ echo Efficient Linux! | rev
!xuniL tneiciffE
```

[1] Вопрос к чит телям: что дел ет конвейер `rev myfile | tac | rev | tac`?

Помимо очевидной р звлек тельной ценности, `rev` удобн для извлечения сложной информ ции из ф йлов. Предположим, у в с есть ф йл с имен ми зн менитостей:

```
$ cat celebrities
Jamie Lee Curtis
Zooey Deschanel
Zendaya Maree Stoermer Coleman
Rihanna
```

и вы хотите вывести н экр н последнее слово из к ждой строки (Curtis, Deschanel, Coleman, Rihanna). Это было бы легко сдел ть с помощью ком нды `cut -f`, если бы в к ждой строке было один ковое количество полей, но это число р злич ется. С помощью `rev` вы можете перевернуть все строки, вырез ть первое слово и снов перевернуть, чтобы добиться жел емого:

```
$ rev celebrities
sitruC eeL eimaJ
lenahcseD yeooZ
nameloC remreotS eeraM ayadneZ
annahiR
$ rev celebrities | cut -d' ' -f1
sitruC
lenahcseD
nameloC
annahiR
$ rev celebrities | cut -d' ' -f1 | rev
Curtis
Deschanel
Coleman
Rihanna
```

Команды awk и sed

`awk` и `sed` — это универс льные «суперком нды» для обр ботки текст . Они позволяют реш ть пр ктически все з д чи, которые р ссм трив лись р нее в этой гл ве, но с помощью более сложного синт ксис . Н пример, они могут вывести первые 10 строк ф йл , к к это дел ет `head`:

```
$ sed 10q myfile          Выводит 10 строк и завершается
$ awk 'FNR<=10' myfile    Выводит строки на экран, до тех пор пока их количество
                          не достигнет 10
```

Их возможности шире всех р нее изученных н ми ком нд для р боты с текстовыми ф йл ми. Н пример, они позволяют з менять или менять мест ми строки:

```
$ echo image.jpg | sed 's/\.jpg/.png/'              Заменить .jpg на .png
image.png
$ echo "linux efficient" | awk '{print $2, $1}'     Поменять местами два слова
efficient linux
```

awk и sed сложнее изучить, чем опис нные выше ком нды, потому что к жд я из них имеет встроенный мини тюрный язык прогр ммиров ния. Об их возможностях н пис ны книги[1]. Я н стоятельно рекомендую потр тить время н изучение обеих ком нд или хотя бы одной из них.

Чтобы н ч ть путешествие в мир awk и sed, р ссмотрим основные принципы к ждой ком нды и некоторые р спростр ненные в ри нты их использов ния. Я т кже порекомендую несколько онл йн-руководств.

Не беспокойтесь о том, чтобы з помнить к ждую функцию awk или sed. Действительно успешное использов ние этих ком нд возможно при двух условиях:

- Если вы поним ете их возможности. Тогд в нужный момент вы ск жете себе: «Аг ! Вот эту р боту сможет сдел ть awk (или sed)!»
- Если н учитесь использов ть спр вочные стр ницы ком нд и н ходить готовые решения н Stack Exchange (https://oreil.ly/0948M) и других онл йн-ресурс х.

Основы awk

awk преобр зует строки текст из ф йлов (или из ст нд ртного ввод) в любой другой текст, используя последов тельность инструкций, которую н зыв ют *awk-прогр ммой*[2]. Чем больше опыт н пис ния awk-прогр мм вы н к плив ете, тем гибче можете м нипулиров ть текстом. Вы можете ук з ть т кую прогр мму в ком ндной строке:

```
$ awk program input-files
```

но т кже можете хр нить одну или несколько awk-прогр мм в ф йл х и обр щ ться к ним с п р метром -f. Прогр ммы буду з пуск ться последов тельно:

```
$ awk -f program-file1 -f program-file2 -f program-file3 input-files
```

Прогр мм awk включ ет одно или несколько действий, т ких к к вычисление зн чений или печ ть текст , которые з пуск ются, когд входн я строк соответствует ш блону. К жд я инструкция в прогр мме имеет вид:

```
шаблон {действие}
```

---

[1] Н пример, «sed & awk», изд нн я O'Reilly.

[2] awk является ббреви турой от ф милий созд телей прогр ммы: Ахо (Aho), В йнбергер (Weinberger) и Керниг н (Kernighan).

Типичные ш блоны включ ют:

.Слово `BEGIN` — действие з пуск ется только один р з, перед обр боткой ввод ком нды `awk`.

.Слово `END` — действие з пуск ется только один р з, после обр ботки ввод ком нды `awk`.

.Регулярное выр жение (см. т бл. 5.1), окруженное слеш ми. Н пример, `/^[A-Z]/` соответствует строк м, н чин ющимся с з гл вной буквы.

П р примеров других выр жений, специфичных для `awk`: `$3~/^[A-Z]/` проверяет, н чин ется ли третье поле в строке ввод (`$3`) с з гл вной буквы; `FNR>5` ук зыв ет `awk` пропустить первые пять строк ввод .

Действие без ш блон выполняется для к ждой строки ввод (несколько awk-прогр мм в р зделе «Ком нд awk {print}» н с. 101 относятся к этому типу). Н пример, `awk` элег нтно реш ет з д чу «н печ т ть ф милию зн менитости» из пример в р зделе «Ком нды rev» н с. 107, н прямую печ т я последнее слово из к ждой строки:

```
$ awk '{print $NF}' celebrities
Curtis
Deschanel
Coleman
Rihanna
```

При вводе awk-прогр ммы в ком ндной строке з ключ йте ее в к вычки, чтобы оболочк не вычислял специ льные символы `awk`. При необходимости используйте один рные или двойные к вычки.

Ш блон без ук з ния действия з пуск ет действие по умолч нию `{print}`, которое просто печ т ет любые совп д ющие входные строки без изменений:

```
$ echo efficient linux | awk '/efficient/'
efficient linux
```

Для более полной демонстр ции д в йте обр бот ем р зделенный т буляцией текст ф йл *animals.txt* из пример 1.1 н с. 21, чтобы созд ть ккур тную библиогр фию. Необходимо преобр зов ть строки вид

```
python Programming Python 2010 Lutz, Mark
```

в следующий форм т:

```
Lutz, Mark (2010). "Programming Python"
```

Необходим перест новк трех полей и доб вление некоторых символов, т ких к к круглые скобки и двойные к вычки. Следующ я awk-прогр мм выполняет это, используя п р метр `-F` для изменения р зделителя ввод с пробелов н т буляцию (`\t`):

```
$ awk -F'\t' '{print $4, "(" $3 ").", "\"" $2 "\""}' animals.txt
Lutz, Mark (2010). "Programming Python"
Barrett, Daniel (2005). "SSH, The Secure Shell"
Schwartz, Randal (2012). "Intermediate Perl"
Bell, Charles (2014). "MySQL High Availability"
Siever, Ellen (2009). "Linux in a Nutshell"
Boney, James (2005). "Cisco IOS in a Nutshell"
Roman, Steven (1999). "Writing Word Macros"
```

Доб вим регулярное выр жение для обр ботки только н зв ния книги, содерж щего *horse*:

```
$ awk -F'\t' '/^horse/ {print $4, "(" $3 ").", "\"" $2 "\""}' animals.txt
Siever, Ellen (2009). "Linux in a Nutshell"
```

Или отберем только книги, изд нные не р ньше 2010 год , проверив, соответствует ли поле `$3` ш блону `^201`:

```
$ awk -F'\t' '$3~/^201/ {print $4, "(" $3 ").", "\"" $2 "\""}' animals.txt
Lutz, Mark (2010). "Programming Python"
Schwartz, Randal (2012). "Intermediate Perl"
Bell, Charles (2014). "MySQL High Availability"
```

Н конец, доб вим инструкцию `BEGIN`, чтобы н печ т ть понятный з головок, дефисы для отступов и инструкцию `END`, чтобы н пр вить чит теля к дополнительной информ ции:

```
$ awk -F'\t' \
  ' BEGIN {print "Recent books:"} \
  $3~/^201/{print "-", $4, "(" $3 ").", "\"" $2 "\""} \
  END {print "For more books, search the web"} ' \
  animals.txt
Recent books:
- Lutz, Mark (2010). "Programming Python"
- Schwartz, Randal (2012). "Intermediate Perl"
- Bell, Charles (2014). "MySQL High Availability"
For more books, search the web
```

Ком нд `awk` умеет гор здо больше, чем просто вывод д нных — он т кже может выполнять вычисления, н пример суммиров ть числ от 1 до 100:

```
$ seq 1 100 | awk '{s+=$1} END {print s}'
5050
```

Чтобы изучить awk, воспользуйтесь учебными пособиями н tutorialspoint.com/awk или riptutorial.com/awk либо выполните поиск в интернете по з просу *awk tutorial*. Результ т в м понр вится.

Улучшенный способ обнаружения дубликатов файлов

В р зделе «Обн ружение дублик тов ф йлов» н с. 32 вы построили конвейер, который обн ружив ет и подсчитыв ет дублик ты ф йлов JPEG по контрольной сумме, но его возможностей не хв т ет для вывод имен ф йлов:

```
$ md5sum *.jpg | cut -c1-32 | sort | uniq -c | sort -nr | grep -v "    1 "
      3 f6464ed766daca87ba407aede21c8fcc
      2 c7978522c58425f6af3f095ef1de1cd5
      2 146b163929b6533f02e91bdf21cb9563
```

Теперь, когд мы зн ем ком нду awk, у н с т кже есть инструменты для печ ти имен ф йлов. Д в йте созд дим новую ком нду, котор я считыв ет к ждую строку вывод md5sum:

```
$ md5sum *.jpg
146b163929b6533f02e91bdf21cb9563 image001.jpg
63da88b3ddde0843c94269638dfa6958 image002.jpg
146b163929b6533f02e91bdf21cb9563 image003.jpg
⋮
```

но не только подсчитыв ет вхождения к ждой контрольной суммы, но и сохр няет имен ф йлов для вывод н экр н. Н м пон добятся две дополнительные возможности ком нды awk — *м ссивы* и *циклы*.

М ссив — это переменн я, содерж щ я н бор зн чений. Если м ссив н зыв ется A и содержит семь зн чений, то к ним можно обр щ ться через *элементы* м ссив A[1], A[2], A[3], вплоть до A[7]. Зн чения от 1 до 7 н зыв ются *ключ ми* м ссив . Вы можете созд ть любые ключи, к кие з хотите. Если вы предпочит ете обр щ ться к элемент м м ссив , используя имен персон жей Диснея, н зовите их A["Doc"], A["Grumpy"], A["Bashful"], вплоть до A ["Dopey"].

Чтобы подсчит ть повторяющиеся изобр жения, созд дим м ссив counts с одним элементом для к ждой контрольной суммы. К ждый ключ м ссив предст вляет собой контрольную сумму, связ нный с ним элемент содержит количество р з, которое контрольн я сумм встреч ется во входных д нных. Н пример, элемент м ссив counts["f6464ed766daca87ba407aede21c8fcc"] может иметь зн чение 3. Следующий сцен рий ком нды awk проверяет к ждую строку вывод md5sum, выделяет контрольную сумму ($1) и использует ее в к честве ключ для м ссив

counts. Оператор ++ увеличивает элемент на 1 каждый раз, когда awk встречает связанную с ним контрольную сумму:

```
$ md5sum *.jpg | awk '{counts[$1]++}'
```

Пока что awk-программа ничего не выводит, он просто обрабатывает каждую контрольную сумму и завершает работу. Чтобы вывести на экран количество посчитанных значений, нам понадобится вторая функция awk, назначаемая циклом for. Цикл for проходит по всем ключам массива и последовательно обрабатывает каждый его элемент. Например, следующая команда выводит значения элементов массива counts по их ключам:

```
for (key in counts) print counts[key]
```

Поместим этот цикл в инструкцию END, чтобы он выполнялся после вычисления всех элементов массива:

```
$ md5sum *.jpg \
  | awk '{counts[$1]++} \
         END { for (key in counts) print counts[key] }'
1
2
2
⋮
```

Затем добавим контрольные суммы в вывод. Каждый ключ массива является контрольной суммой, поэтому просто выведем их:

```
$ md5sum *.jpg \
  | awk '{counts[$1]++} \
         END {for (key in counts) print counts[key] " " key }'
1 714eceeb06b43c03fe20eb96474f69b8
2 146b163929b6533f02e91bdf21cb9563
2 c7978522c58425f6af3f095ef1de1cd5
⋮
```

Для сбора и вывода имен файлов используем массив names также с контрольными суммами в качестве ключей. Так как в awk обрабатывается каждую строку вывода, добавим имя файла ($2) к соответствующему элементу массива имен вместе с пробелом в качестве разделителя. В цикле END после печати контрольной суммы (key) выведем двоеточие и собранные имена файлов для этой контрольной суммы:

```
$ md5sum *.jpg \
  | awk '{counts[$1]++; names[$1]=names[$1] " " $2 } \
         END {for (key in counts) print counts[key] " " key ":" names[key] }'
1 714eceeb06b43c03fe20eb96474f69b8: image011.jpg
2 146b163929b6533f02e91bdf21cb9563: image001.jpg image003.jpg
2 c7978522c58425f6af3f095ef1de1cd5: image019.jpg image020.jpg
⋮
```

Строки, н чин ющиеся с единицы, предст вляют собой контрольные суммы, которые встреч ются только один р з, то есть не дублируются. Н пр вим вывод в `grep -v`, чтобы уд лить эти строки, з тем отсортируем результ ты от большего к меньшему с помощью `sort -nr` — и получим жел емый результ т:

```
$ md5sum *.jpg \
  | awk '{counts[$1]++; names[$1]=names[$1] " " $2} \
         END {for (key in counts) print counts[key] " " key ":" names[key]}' \
  | grep -v '^1 ' \
  | sort -nr
3 f6464ed766daca87ba407aede21c8fcc: image007.jpg image012.jpg image014.jpg
2 c7978522c58425f6af3f095ef1de1cd5: image019.jpg image020.jpg
2 146b163929b6533f02e91bdf21cb9563: image001.jpg image003.jpg
```

Основы sed

Ком нд `sed` преобр зует текст из ф йлов или из ст нд ртного ввод , используя последов тельность инструкций, которую н зыв ют *sed-сцен рием*[1]. Сцен рии `sed` н первый взгляд м лопонятны. Примером может служить `s/Windows/Linux/g`, з меняющий к ждое вхождение строки *Windows* н *Linux*. Термин *сцен рий* в д нном случ е озн ч ет не ф йл (н пример, сцен рий оболочки), строку[2]. Вызовите сцен рий `sed` в ком ндной строке:

```
$ sed script input-files
```

или используйте п р метр `-e` для поддержки нескольких сцен риев, которые последов тельно обр б тыв ют ввод:

```
$ sed -e script1 -e script2 -e script3 input-files
```

Т кже можно хр нить sed-сцен рии в ф йл х и обр щ ться к ним с п р метром `-f`, тогд они будут з пуск ться последов тельно:

```
$ sed -f script-file1 -f script-file2 -f script-file3 input-files
```

К к и в случ е с `awk`, преимуществ использов ния `sed` з висят от в шего умения созд в ть сцен рии. Н иболее ч сто используемым сцен рием является подст новк , котор я з меняет одни строки другими. Ее синт ксис:

```
s/regexp/replacement/
```

---

[1] Н зв ние ком нды `sed` является сокр щением от *stream editor*, потому что он ред ктирует текстовый поток.

[2] Если вы зн комы с ред ктор ми `vi`, `vim`, `ex` или `ed`, синт ксис сцен рия `sed` может пок з ться в м зн комым.

где *regexp* — регулярное выр жение для сопост вления с к ждой входной строкой (см. т бл. 5.1); *replacement* — строк для з мены совп д ющего текст . В к честве простого пример з мен слов:

```
$ echo Efficient Windows | sed "s/Windows/Linux/"
Efficient Linux
```

При вводе sed-сцен рия в ком ндной строке з ключ йте его в к вычки, чтобы оболочк не вычислял специ льные символы `sed`. При необходимости используйте один рные или двойные к вычки.

`sed` легко реш ет з д чу вывод ф милии зн менитости из р здел «Ком нд rev» н с. 107 с помощью регулярного выр жения. Просто сопост вьте все символы (`.*`) до последнего пробел и ничем их не з меняйте:

```
$ sed 's/.* //' celebrities
Curtis
Deschanel
Coleman
Rihanna
```

Подстановка и символ слеша

Кос я черт в подст новке может быть з менен любым другим удобным символом. Это полезно, когд с мо регулярное выр жение включ ет слеш, который в противном случ е пришлось бы экр ниров ть. Следующие три сцен рия sed эквив лентны:

```
s/one/two/
s_one_two_
s@one@two@
```

В сцен рии после подст новки могут следов ть несколько п р метров, влияющих н результ т. Н пример, п р метр `i` дел ет совп дения нечувствительными к регистру:

```
$ echo Efficient Stuff | sed "s/stuff/linux/"          Чувствительно к регистру. Нет
                                                       совпадений
Efficient Stuff
$ echo Efficient Stuff | sed "s/stuff/linux/i"         Нечувствительно к регистру
Efficient linux
```

П р метр `g` (*global*) з меняет все вхождения регулярного выр жения, не только первое:

```
$ echo efficient stuff | sed "s/f/F/"          Заменяет только первую «f»
eFficient stuff
$ echo efficient stuff | sed "s/f/F/g"         Заменяет все вхождения «f»
eFFicient stuFF
```

Другим р спростр ненным типом sed-сцен рия является уд ление. Сцен рий, уд ляющий строки по их номеру:

```
$ seq 10 14 | sed 4d                           Удаляет четвертую строку
10
11
12
14
```

Сцен рий, уд ляющий строки, которые соответствуют регулярному выр жению:

```
$ seq 101 200 | sed '/[13579]$/d'              Удаляет строки, заканчивающиеся
                                               на нечетные цифры
102
104
106
⋮
200
```

Сопоставление подвыражений с sed

Предположим, у н с есть несколько имен ф йлов:

```
$ ls
image.jpg.1 image.jpg.2 image.jpg.3
```

и мы хотим созд ть новые имен , *image1.jpg*, *image2.jpg* и *image3.jpg*. `sed` может р збить имен ф йлов н ч сти и изменить их порядок с помощью функции, н зыв емой *подвыр жениями*. Сн ч л созд дим регулярное выр жение, соответствующее имен м ф йлов:

```
image\.jpg\.[1-3]
```

Чтобы переместить последнюю цифру в имени ф йл н другую позицию, изолируем ее, окружив символ ми `\(` и `\)`. Это определяет подвыр жение — выделенную ч сть регулярного выр жения:

```
image\.jpg\.  \(  [1-3]  \)
```

`sed` может ссыл ться н подвыр жения по номеру и упр влять ими. Мы созд ли только одно подвыр жение, поэтому его имя `\1`. Вторым подвыр жением будет

\2 и т. д., м ксимум — \9. Новые имен ф йлов будут иметь вид image\1.jpg. Следов тельно, sed-сцен рий будет т ким:

```
$ ls | sed "s/image\.jpg\.\([1-3]\)/image\1.jpg/"
image1.jpg
image2.jpg
image3.jpg
```

Чтобы усложнить ситу цию, предположим, что имен ф йлов имеют больше отличий и состоят из слов нижнего регистр :

```
$ ls
apple.jpg.1 banana.png.2 carrot.jpg.3
```

Созд дим три подвыр жения для з хв т н ч льного имени ф йл , р сширения и последней цифры:

```
\([a-z][a-z]*\)          \1 = Начальное имя файла из одной буквы или более
\([a-z][a-z][a-z]\)      \2 = Расширение файла из трех букв
\([0-9]\)                \3 = Цифра
```

Соединим их с помощью экр ниров нных точек (\.), чтобы сформиров ть следующее регулярное выр жение:

`\([a-z][a-z]*\)` \. `\([a-z][a-z][a-z]\)` \. `\([0-9]\)`

Преобр зуем имен ф йлов в форм те sed к к \1\3.\2, тогд оконч тельный сцен рий будет выглядеть т к:

```
$ ls | sed "s/\([a-z][a-z]*\)\.\([a-z][a-z][a-z]\)\.\([0-9]\)/\1\3.\2/"
apple1.jpg
banana2.png
carrot3.jpg
```

Эт ком нд не переименовыв ет ф йлы, он просто выводит н экр н новые имен . В р зделе «Вст вк имени ф йл в последов тельность» н с. 169 пок з н н логичный пример, в котором т кже выполняется переименов ние.

Чтобы изучить sed, воспользуйтесь учебник ми https://tutorialspoint.com/sed или https://grymoire.com/Unix/Sed.html либо выполните поиск в интернете по з просу *sed tutorial*.

Как расширить инструментарий

Большинство систем Linux пост вляются с тысяч ми прогр мм ком ндной строки, и большинство из них имеют множество п р метров, изменяющих их

поведение. Вы вряд ли это все выучите и з помните. Отсюд возник ет вопрос: к к н йти нужную прогр мму или д птиров ть другую, которую вы уже зн ете, для достижения своих целей?

Первый и с мый очевидный ш г — поиск в интернете. Н пример, если в м нужн ком нд , котор я огр ничив ет ширину строк в текстовом ф йле, перенося слишком длинные строки, поищите по фр зе «перенос строк ком нд Linux» («Linux command wrap lines») — и вы увидите ком нду `fold`:

```
$ cat title.txt
This book is titled "Efficient Linux at the Command Line"
$ fold -w40 title.txt
This book is titled "Efficient Linux at
the Command Line"
```

Чтобы н йти ком нды, которые уже уст новлены в в шей системе Linux, з пустите `man -k` (или, что то же с мое, ком нду `apropos`). Получив слово, `man -k` ищет его в кр тких опис ниях спр вочных стр ниц:

```
$ man -k width
DisplayWidth (3) - image format functions and macros
DisplayWidthMM (3) - image format functions and macros
fold (1) - wrap each input line to fit in specified width
⋮
```

`man -k` р бот ет с регулярными выр жениями в стиле `awk` в строк х поиск (см. т блицу 5.1):

```
$ man -k "wide|width"
```

Ком нд , котор я отсутствует в в шей системе, может быть уст новлен через менеджер п кетов в шей версии Linux. Менеджер п кетов — это прогр ммное обеспечение для уст новки прогр мм Linux, которые поддержив ются в шей системой. Популярными менеджер ми п кетов являются `apt`, `dnf`, `emerge`, `pacman`, `rpm`, `yum` и `zypper`. Используйте ком нду `man`, чтобы выяснить, к кой менеджер п кетов уст новлен в в шей системе, и узн йте, к к иск ть неуст новленные п кеты. Ч сто требуется последов тельность из двух ком нд: первой — для копиров ния последних д нных о доступных п кет х (мет д нных) из интернет в в шу систему, второй — для поиск мет д нных. Н пример, для систем н б зе *Ubuntu* или *Debian Linux* ком нды следующие:

```
$ sudo apt update                  Скачать новейшие метаданные
$ apt-file search string           Искать строку string
```

Если после долгих поисков вы не н шли ком нду, отвеч ющую в шим з д ч м, обр титесь з помощью н онл йн-форум. Отличной отпр вной точкой, чтобы з д в ть пр вильные вопросы, является ст тья *«How do I ask a good question?»*

н форуме *Stack Overflow* (https://stackoverflow.com/help/how-to-ask). Формулируйте свои вопросы, ув ж я время других людей, тогд более опытные пользов тели будут охотнее н них отвеч ть. В ш вопрос должен быть кр тким и по существу, включ ть сообщения об ошибк х или другие выходные д нные и объяснять, что вы уже пробов ли сдел ть с мостоятельно. Потр тив время, чтобы сформулиров ть к чественный вопрос, вы увеличите ш нсы н полезный ответ не только для себя, но и для других людей, которые столкнулись со схожей проблемой.

Резюме

Теперь вы вышли з р мки инструмент рия р змером с ч йную ложку из гл вы 1 и готовы реш ть сложные з д чи. Следующие гл вы н полнены пр ктическими пример ми использов ния ком нд, которые вы узн ли, в с мых р зных ситу циях.

ГЛАВА 6

Родители, потомки и окружение

Н зн чение оболочки — выполнение ком нд — н столько фунд мент льно, что можно подум ть, что оболочк встроен в Linux к ким-то особым обр зом. Но это не т к. Оболочк — это обычн я прогр мм , т к я к к `ls` или `cat`. Он з прогр ммиров н н повторение следующих ш гов снов и снов , снов и снов ...

1. Вывести пригл шение ком ндной строки.
2. Прочит ть ком нду из ст нд ртного ввод .
3. Интерпретиров ть и з пустить ком нду.

Linux прекр сно скрыв ет тот ф кт, что оболочк — это обычн я прогр мм . Когд вы входите в систему, Linux втом тически з пуск ет для в с экземпляр оболочки, известный к к *ком ндн я оболочк вход в систему* (*login shell*). Он з пуск ется т к пл вно, что к жется , будто *это и есть с м* Linux, хотя н с мом деле это просто прогр мм , з пущенн я от в шего имени для вз имодействия с Linux.

Где находится ваша оболочка входа?

Если вы входите в систему без гр фического интерфейс , ск жем, с помощью клиентской прогр ммы SSH, оболочк вход в систему является н ч льной оболочкой, с которой вы вз имодействуете. Он печ т ет первое пригл шение и ожид ет в шей ком нды.

Если же вы используете версию с гр фическим интерфейсом, в ш оболочк вход в систему выполняется нез метно для в с. Он з пуск ет среду р бочего стол , т кую к к GNOME, Unity, Cinnamon или KDE Plasma. З тем вы можете открыть окн термин л для з пуск дополнительных интер ктивных экземпляров оболочки.

Чем больше вы поним ете в устройстве оболочки, тем эффективнее сможете р бот ть с Linux и тем меньше «м гии» ост нется. В этой гл ве более дет льно, чем в гл ве 2, р ссмотрены следующие з г дки оболочки:

- Где н ходятся прогр ммы оболочки.
- К к р зные экземпляры оболочки могут быть связ ны друг с другом.
- Почему р зные экземпляры оболочки могут иметь одни и те же переменные, зн чения, псевдонимы и другой контекст.
- К к изменить поведение оболочки по умолч нию, отред ктиров в ф йлы конфигур ции.

К концу гл вы вы поймете, что все эти т йны не т кие уж и з г дочные.

Оболочки — это исполняемые файлы

Оболочкой по умолч нию в большинстве систем Linux является `bash`[1], и это обычн я прогр мм — исполняемый ф йл, р сположенный в системном к т логе */bin* вместе с `cat`, `ls`, `grep` и другими зн комыми ком нд ми:

```
$ cd /bin
$ ls -l bash cat ls grep
-rwxr-xr-x 1 root root 1113504 Jun 6 2019 bash
-rwxr-xr-x 1 root root 35064 Jan 18 2018 cat
-rwxr-xr-x 1 root root 219456 Sep 18 2019 grep
-rwxr-xr-x 1 root root 133792 Jan 18 2018 ls
```

Скорее всего, `bash` — не единственн я возможн я оболочк в в шей системе. Допустимые оболочки обычно перечислены в ф йле */etc/shells*:

```
$ cat /etc/shells
/bin/sh
/bin/bash
/bin/csh
/bin/zsh
```

Чтобы узн ть, к кую оболочку вы используете, примените ком нду `echo` к переменной оболочки `SHELL`:

```
$ echo $SHELL
/bin/bash
```

Теоретически систем Linux может р ссм трив ть любую прогр мму к к допустимую оболочку вход , если учетн я з пись пользов теля н строен н вызов ее при входе в систему и он ук з н в */etc/shells* (если это требуется в в шей системе). Обл д я привилегиями суперпользов теля, вы д же можете н пис ть и уст новить свою собственную оболочку, т кую, н пример, к к в листинге 6.1.

[1] Если вы используете другую оболочку, обратите внимание на Приложение Б.

Он чит ет любую ком нду и отвеч ет: «Извините, боюсь, я не могу этого сдел ть». Эт пользов тельск я оболочк н меренно сдел н глупой, но он демонстрирует, что другие прогр ммы могут быть т кой же легитимной оболочкой, к к */bin/bash*.

Листинг 6.1. `halshell`: оболочка, которая отказывается выполнять ваши команды

```
#!/bin/bash
# Вывод приглашения командной строки
echo -n '$ '
# Чтение ввода пользователя в цикле. Выход, когда пользователь нажимает Ctrl-D
while read line; do
# Игнорировать входную строку $line и вывести сообщение
echo "Извините, боюсь, я не могу этого сделать"
# Вывод следующего приглашения командной строки
echo -n '$ '
done
```

Поскольку `bash` — это просто прогр мм , вы можете з пустить ее вручную, к к и любую другую ком нду:

```
$ bash
```

Если вы это сдел ете, вы просто увидите еще одно пригл шение, к к будто в ш ком нд не имел ник кого эффект :

```
$
```

Но н с мом деле вы з пустили новый экземпляр `bash`. Этот новый экземпляр печ т ет пригл шение и ожид ет в шей ком нды. Чтобы к к-то выделить новый экземпляр, измените его пригл шение ком ндной строки (ск жем, н `%%`) с помощью переменной оболочки `PS1` и выполните несколько ком нд:

```
$ PS1="%% "
%% ls                          Приглашение изменилось
animals.txt
%% echo "This is a new shell"
This is a new shell
```

Теперь з пустите ком нду `exit`, чтобы з вершить р боту нового экземпляр `bash`. Вы вернетесь к исходной оболочке, в которой используется пригл шение со зн ком долл р :

```
%% exit
$
```

Необходимо отметить, что изменение с `%%` обр тно н `$` не было просто изменением пригл шения ком ндной строки. Это был полн я смен оболочки. Р бот

нового экземпляр bash з вершен , поэтому исходн я оболочк з пр шив ет следующую ком нду.

З пуск bash вручную нужен не только для р звлечения. Вы будете использов ть вызыв емые вручную экземпляры оболочки в гл ве 7.

Родительский и дочерний процессы

Когд один экземпляр оболочки вызыв ет другой, к к только что было пок з но, исходн я оболочк н зыв ется *родительской*, новый экземпляр — *дочерней*. То же с мое верно для любой прогр ммы, котор я вызыв ет другую прогр мму Linux. Вызыв ющ я прогр мм является родительской, вызыв ем я — дочерней. Р бот ющ я прогр мм Linux н зыв ется *процессом*, поэтому вы т кже встретите термины *родительский процесс* и *дочерний процесс,* или *потомок*. Процесс может иметь любое количество потомков, но у к ждого потомк есть только один родитель.

У к ждого процесс есть свое окружение. Окружение, которое вы, возможно, помните из р здел «Окружение и ф йлы иници лиз ции, кр тк я версия» н с. 50, включ ет в себя текущий к т лог, путь поиск , пригл шение оболочки и другую в жную информ цию, хр нящуюся в переменных оболочки. Когд созд ется дочерний элемент, его окружение в зн чительной степени является копией окружения его родителя (подробности см. в р зделе «Переменные окружения» н с. 125).

К ждый р з, когд вы з пуск ете простую ком нду, вы созд ете дочерний процесс. Это н столько в жный момент для поним ния Linux, что повторим еще р з: д же когд вы з пуск ете простую ком нду, т кую к к ls, он выполняется внутри нового дочернего процесс со своим собственным (скопиров нным) окружением. Это озн ч ет, что любые изменения, которые вы вносите в дочерний процесс, н пример изменение переменной пригл шения PS1 в дочерней оболочке, влияют только н дочерний процесс и теряются при выходе из него. Точно т к же любые изменения в родительском элементе не повлияют н его дочерние элементы, которые уже з пущены. Одн ко изменения в родительском объекте *могут* повлиять н его *будущие* дочерние элементы, поскольку окружение к ждого дочернего объект копируется при з пуске из родительского.

Почему в жно, чтобы ком нды выполнялись в дочерних процесс х? Прежде всего, люб я прогр мм , которую вы з пуск ете, может выполнять ком нду cd по всей ф йловой системе, но, когд он з верш ется, в ш текущ я оболочк (родительск я) не меняет свой текущий к т лог. Проведем эксперимент, чтобы док з ть это. Созд йте в своем дом шнем к т логе небольшой сцен рий с именем cdtest, содерж щий ком нду cd:

```
#!/bin/bash
cd /etc
echo "Here is my current directory:"
pwd
```

Сделайте его исполняемым:

```
$ chmod +x cdtest
```

Выведите имя текущего каталога и запустите скрипт:

```
$ pwd
/home/smith
$ ./cdtest
Here is my current directory:
/etc
```

Теперь проверьте вашу текущий каталог:

```
$ pwd
/home/smith
```

Ваш текущий каталог не изменился, хотя скрипт `cdtest` переместился в каталог */etc*. Это потому, что `cdtest` запускается внутри дочернего процесса со своим собственным окружением. Изменения в дочернем окружении не могут повлиять на родительское, поэтому текущий каталог родителя не изменился. То же самое происходит, когда вы запускаете исполняемую программу, такую как `cat` или `grep`, — она запускается в дочернем процессе, который завершается после окончания работы программы, забирая с собой любые изменения окружения.

Почему команда cd должна быть встроенной в оболочку

Если программы Linux не могут изменить текущий каталог вашей оболочки, то как команда `cd` может изменить его? Оказывается, `cd` — это не программа, встроенная функция оболочки (*shell builtin*). Если бы `cd` была внешней по отношению к оболочке программой, изменения каталога были бы невозможны, так как они выполнялись бы в дочернем процессе и не могли бы повлиять на родительский процесс.

Конвейеры запускают несколько дочерних процессов: по одному для каждой команды в конвейере. Например, эта команда из раздела «Команда 6: `uniq`» на с. 30 запускает шесть потомков:

```
$ cut -f1 grades | sort | uniq -c | sort -nr | head -n1 | cut -c9
```

Переменные окружения

К ждый экземпляр оболочки имеет свой н бор переменных, к к вы узн ли из р здел «Вычисление переменных» н с. 40. Некоторые переменные существуют только в одной конкретной оболочке. Они н зыв ются *лок льными переменными*. Другие переменные втом тически копируются из оболочки во все ее дочерние элементы. Это *переменные окружения*, и все вместе они формируют окружение оболочки.

Некоторые примеры переменных окружения и их использов ния.

`HOME` — путь к в шему дом шнему к т логу. Его зн чение втом тически уст - н влив ется оболочкой при входе в систему. Текстовые ред кторы, т кие к к `vim` и `emacs`, чит ют переменную `HOME`, чтобы н йти свои ф йлы конфигур ции (*$HOME/.vim* и *$HOME/.emacs* соответственно).

`PWD` — текущий к т лог в шей оболочки. Его зн чение втом тически уст н влив ется и поддержив ется оболочкой к ждый р з, когд вы переходите в другой к т лог с помощью ком нды `cd`. Ком нд `pwd` считыв ет переменную `PWD`, чтобы вывести имя текущего к т лог в шей оболочки.

`EDITOR` — имя или путь предпочит емого в ми текстового ред ктор . Его зн чение обычно з д ется в ф йле конфигур ции оболочки. Другие прогр ммы чит ют эту переменную, чтобы з пустить соответствующий ред ктор от в шего имени.

Просмотрите переменные окружения в шей оболочки с помощью ком нды `printenv`. Вывод предст вляет собой несортиров нный н бор строк — по одной переменной в строке, и может быть довольно длинным, поэтому используйте конвейер из `sort` и `less` для более удобного просмотр [1]:

```
$ printenv | sort -i | less
⋮
DISPLAY=:0
EDITOR=emacs
HOME=/home/smith
LANG=en_US.UTF-8
PWD=/home/smith/Music
SHELL=/bin/bash
TERM=xterm-256color
USER=smith
⋮
```

[1] Я сократил вывод, оставив только самые общие переменные оболочки. Ваш вывод, вероятно, намного длиннее и полон неясных имен переменных.

Лок льные переменные не отобр ж ются в выводе `printenv`. Проверьте их зн чения, пост вив перед именем переменной зн к долл р , с помощью ком нды `echo`:

```
$ title="Efficient Linux"
$ echo $title
Efficient Linux
$ printenv title                (пустой вывод)
```

Создание переменных окружения

Чтобы превр тить лок льную переменную в переменную окружения, используйте ком нду `export`:

```
$ MY_VARIABLE=10                Присвоим значение локальной переменной
$ export MY_VARIABLE            Экспортируем, чтобы она стала переменной окружения
$ export ANOTHER_VARIABLE=20    Все вместе в одной команде
```

`export` озн ч ет, что переменн я и ее зн чение будут скопиров ны из текущей оболочки во всех ее будущих потомков. Лок льные переменные не копируются в будущие дочерние элементы:

```
$ export E="I am an environment variable"   Установить переменную оболочки
$ L="I am just a local variable"            Установить локальную переменную
$ echo $E
I am an environment variable
$ echo $L
I am just a local variable
$ bash                                      Запустить дочернюю оболочку
$ echo $E                                   Переменная оболочки была скопирована
I am an environment variable
$ echo $L                                   Локальная переменная не была скопирована
                                            Вывод пустой строки
$ exit                                      Выход из дочернего процесса
```

Помните, что дочерние переменные — это *копии*. Любые изменения в копиях не влияют н родительскую оболочку:

```
$ export E="I am the original value"   Установить переменную оболочки
$ bash                                 Запустить дочернюю оболочку
$ echo $E
I am the original value                Родительское значение было скопировано
$ E="I was modified in a child"        Изменить дочернюю копию
$ echo $E
I was modified in a child
$ exit                                 Выход из дочерней оболочки
$ echo $E
I am the original value                Родительское значение переменной осталось
                                       без изменений
```

З пустите новую оболочку и измените что-либо в ее окружении, но все изменения исчезнут, когд вы выйдете из оболочки. Это озн ч ет, что вы можете безоп сно экспериментиров ть с функциями оболочки — просто з пустите оболочку вручную, созд в дочернюю, и з вершите ее, когд з кончите.

Предупреждение о мифе: «глобальные» переменные

Иногд Linux слишком хорошо скрыв ет свою внутреннюю р боту. Отличный пример — поведение переменных окружения. К ким-то обр зом, к к по волшебству, т кие переменные, к к HOME и PATH, имеют постоянное зн чение во всех экземпляр х в шей оболочки. В к ком-то смысле они к жутся «глоб льными переменными» (это утверждение встреч ется в других книг х по Linux, не выходивших в изд тельстве O'Reilly). Но переменн я окружения *не является глоб льной*. К ждый экземпляр оболочки имеет свою собственную копию. Изменение переменной окружения в одной оболочке не может изменить ее зн чение ни в одной другой з пущенной оболочке. Модифик ции влияют только н будущих потомков этой оболочки (еще не вызв нных).

Но к ким обр зом т к я переменн я, к к HOME или PATH, сохр няет свое зн чение во всех экземпляр х в шей оболочки? Для этого есть две причины, которые проиллюстриров ны н рис. 6.1. Если коротко:

Потомки копируют своих родителей.

> Зн чения т ких переменных, к к HOME, обычно уст н влив ются и экспортируются в шей оболочкой вход в систему. Все будущие оболочки (пок вы не выйдете из системы) являются потомк ми оболочки вход в систему, поэтому они получ ют копию переменной и ее зн чение. Т кого род определяемые системой переменные окружения н столько редко изменяются в ре льной р боте, что к жутся глоб льными, но н с мом деле это обычные переменные, подчиняющиеся общим пр вил м (вы д же можете изменить их зн чения в р бот ющей оболочке, но при этом возможно н рушение ожид емого поведения этой ком ндной оболочки и других прогр мм).

Р зные экземпляры оболочки чит ют одни и те же ф йлы конфигур ции.

> Зн чения лок льных переменных, которые не копируются в дочерние элементы, могут быть уст новлены в ф йле конфигур ции Linux, н пример *$HOME/.bashrc* (подробнее см. в р зделе «Н стройк окружения» н с. 129). К ждый экземпляр оболочки при вызове считыв ет и выполняет соответствующие ф йлы конфигур ции. В результ те эти лок льные переменные к жутся копируемыми из оболочки в оболочку. То же с мое относится и к другим неэкспортируемым сущностям оболочки, н пример псевдоним м.

Т кое поведение з ст вляет некоторых пользов телей пол г ть, что ком нд `export` созд ет глоб льную переменную. Это не т к. Ком нд `export WHATEVER` просто объявляет, что переменн я `WHATEVER` будет скопиров н из текущей оболочки во все будущие дочерние элементы.

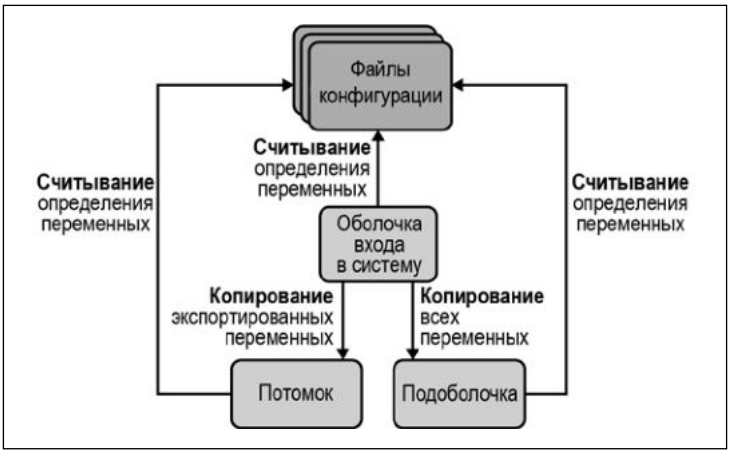


Рис. 6.1. Экземпляры оболочки могут использовать одни и те же значения переменных благодаря экспорту или чтению общих файлов конфигурации

Дочерние оболочки vs подоболочки

Потомок является ч стичной копией своего родителя. Н пример, он включ ет копии переменных окружения своего родителя, но не лок льные (неэкспортиров нные) переменные или псевдонимы родителя:

```
$ alias                 Список псевдонимов
alias gd='pushd'
alias l='ls -CF'
alias pd='popd'
$ bash --norc           Запустить дочернюю оболочку и игнорировать файл .bashrc
$ alias                 Список псевдонимов — ни один не известен
$ echo $HOME            Переменные окружения известны
/home/smith
$ exit                  Выход из дочерней оболочки
```

Если вы когд -нибудь з думыв лись, почему в ши псевдонимы недоступны в сцен риях оболочки, теперь вы зн ете ответ. Сцен рии з пуск ются в дочернем элементе, который не получ ет копии псевдонимов своего родителя.

Подоболочк , н против, является полной копией своего родителя[1]. Он включ ет в себя все родительские переменные, псевдонимы, функции и многое другое. Чтобы з пустить ком нду в подоболочке, з ключите ее в круглые скобки:

```
$ (ls -l)                       Запускает ls -l в подоболочке
-rw-r--r-- 1 smith smith 325 Oct 13 22:19 animals.txt
$ (alias)                       Просмотр псевдонимов в подоболочке
alias gd=pushd
alias l=ls -CF
alias pd=popd
⋮
$ (l)                           Запуск псевдонима от родителя
animals.txt
```

Чтобы проверить, является ли экземпляр оболочки подоболочкой, выведите переменную `BASH_SUBSHELL`. В подоболочк х зн чение будет отлично от нуля:

```
$ echo $BASH_SUBSHELL           Проверяем текущий экземпляр оболочки
0                               Это не подоболочка
$ bash                          Запускаем дочернюю оболочку
$ echo $BASH_SUBSHELL           Проверяем дочернюю оболочку
0                               Это не подоболочка
$ exit                          Выходим из дочерней оболочки
$ (echo $BASH_SUBSHELL)         Явно запускаем подоболочку
1                               Это подоболочка
```

Некоторые пр ктические применения подоболочек мы р ссмотрим в р зделе «Способ 10: Явные подоболочки» н с. 159. А пок просто имейте в виду, что вы можете созд в ть их и они копируют псевдонимы родителя.

Настройка окружения

Когд ком нд `bash` з пуск ется, он н стр ив ет себя, чит я и выполняя последов тельность *ф йлов конфигур ции*. Эти ф йлы определяют переменные, псевдонимы, функции, другие свойств оболочки и могут з пуск ть любую ком нду Linux (они похожи н сцен рии н стройки оболочки). В к т логе */etc* ф йлы конфигур ции определяются системным дминистр тором и применяются ко всем пользов телям системы. Другие ф йлы конфигур ции прин длеж т отдельным пользов телям и изменяются ими. Они р сположены в дом шнем к т логе пользов теля. В т бл. 6.1 перечислены ст нд ртные ф йлы конфигур ции `bash`. Они делятся н нескольких типов:

[1] Копия полн я, з исключением ловушек, которые «сбр сыв ются до зн чений, ун следов нных оболочкой от своего родителя при вызове» (`man bash`). В этой книге мы не будем обсужд ть ловушки.

Ф йлы з пуск

Ф йлы конфигур ции, которые выполняются втом тически при входе в систему, то есть они применяются только к в шей оболочке вход в систему. К примеру, ком нд в т ком ф йле может уст новить и экспортиров ть переменную оболочки. Одн ко определение псевдоним в этом ф йле м лополезно, поскольку псевдонимы не копируются в дочерние элементы.

Ф йлы иници лиз ции

Ф йлы конфигур ции, которые выполняются для к ждого экземпляр оболочки, з исключением оболочки вход в систему, н пример когд вы вручную з пуск ете интер ктивную оболочку или сцен рий неинтер ктивной оболочки. Ком нды ф йл иници лиз ции могут уст новить переменную или определить псевдоним.

Ф йлы очистки при выходе

Ф йлы конфигур ции, которые выполняются непосредственно перед выходом из оболочки вход в систему. Н пример, в т ком ф йле может быть ком нд `clear`, очищ ющ я экр н при выходе из системы.

Таблица 6.1. Стандартные файлы конфигурации, используемые в bash

Тип файла	Чем запускается	Общесистемное местоположение	Расположение личных файлов (в порядке вызова)
Файлы запуска	Оболочка входа в систему (при вызове)	*/etc/profile*	*$HOME/.bash_profile, $HOME/.bash_login, and $HOME/.profile*
Файлы инициализации	Интерактивная оболочка (без авторизации). Сценарии оболочки (при вызове)	*/etc/bash.bashrc* Задайте для переменной BASH_ENV абсолютный путь к файлу инициализации (пример: `BASH_ENV=/usr/local/etc/bashrc`)	*$HOME/.bashrc* Задайте для переменной BASH_ENV абсолютный путь к файлу инициализации (пример: `BASH_ENV=/usr/local/etc/bashrc`)
Файлы очистки	Оболочка входа в систему (при выходе)	*/etc/bash.bash_logout*	*$HOME/.bash_logout*

Обр тите вним ние, что у в с есть три в ри нт ф йлов з пуск в в шем дом шнем к т логе: *.bash_profile*, *.bash_login* и *.profile*. Большинство пользов телей могут выбр ть только один из них. В ш дистрибутив Linux, вероятно,

уже предост вляет один из этих ф йлов предв рительно з полненным полезными ком нд ми. Но, если вы используете не `bash`, , н пример, `Bourne` (*/bin/sh*) или `Korn` (*/bin/ksh*), они чит ют *.profile* и могут д ть сбой, если этот ф йл содержит ком нды, специфичные для `bash`. Поэтому поместите специфичные для `bash` ком нды в *.bash_profile* или *.bash_login* (выберите один из них и используйте только его).

Иногд пользов телей сбив ет с толку возможность использов ния р зных ф йлов з пуск и иници лиз ции. Почему вы хотите, чтобы в ш оболочк вход в систему вел себя ин че, чем другие оболочки, н пример те, которые вы открыв ете в нескольких окн х? К к пр вило, в м не нужно, чтобы они вели себя по-р зному. В ш ф йл з пуск может дел ть немного больше, чем ф йл иници лиз ции *$HOME/.bashrc*, тогд все интер ктивные оболочки (требующие или не требующие вход в систему) будут иметь один ковую конфигур цию.

Когд вы входите в гр фическое окружение р бочего стол (GNOME, Unity, KDE Plasma и т. д.), оболочк вход может быть скрыт . В этом случ е оболочк вход в систему не имеет зн чения, т к к вы вз имодействуете только с ее дочерними элемент ми. Поэтому вы можете поместить большую ч сть или д же всю конфигур цию в ф йл *$HOME/.bashrc*[1]. С другой стороны, если вы входите в систему из негр фической термин льной прогр ммы, т кой, н пример, к к SSH-клиент, то н прямую вз имодействуете со своей оболочкой вход , поэтому ее конфигур ция имеет большое зн чение.

В к ждом из перечисленных случ ев, к к пр вило, имеет смысл, чтобы в ш личный ф йл з пуск имел в к честве источник ф йл иници лиз ции:

```
# Поместите в $HOME/.bash_profile или другой персональный файл запуска
if [ -f "$HOME/.bashrc" ]
then
  source "$HOME/.bashrc"
fi
```

В любом случ е ст р йтесь не помещ ть один ковые ком нды в дв р зных ф йл конфигур ции. Это приводит к пут нице, т кую систему трудно поддержив ть, потому что любое изменение в одном ф йле н до не з быть продублиров ть в другом (и вы обяз тельно з будете, поверьте мне). Вместо этого используйте один ф йл в к честве источник для другого, к к пок з но выше.

[1] Чтобы еще немного з пут ть ситу цию, некоторые окружения р бочего стол имеют свои собственные ф йлы конфигур ции оболочки. Н пример, в GNOME есть *$HOME/.gnomerc*, в б зовой системе X Window — *$HOME/.xinitrc*.

Повторное считывание файла конфигурации

Когд вы изменяете ф йлы з пуск или иници лиз ции, то можете з ст вить р бот ющую оболочку повторно счит ть его, к к опис но в р зделе «Окружения и ф йлы иници лиз ции, кр тк я версия» н с. 50:

```
$ source ~/.bash_profile  Используется встроенная команда source
$ . ~/.bash_profile       Используется точка
```

Зачем нужно считывать файл конфигурации

Почему вы считыв ете ф йл конфигур ции, вместо того чтобы сдел ть его исполняемым с помощью `chmod` и з пустить к к сцен рий? Потому что сцен рий з пуск ется в дочернем процессе. Ком нды в сцен рии не повлияют н в шу родительскую оболочку, после з вершения дочернего процесс все изменения будут потеряны.

Путешествие с вашим окружением

Если вы используете несколько компьютеров с Linux в р зных мест х, в к кой-то момент может пон добиться уст новить отл женные ф йлы конфигур ции более чем н одну м шину. Не копируйте отдельные ф йлы из одной системы в другую — т кой подход в конечном итоге приведет к пут нице. Вместо этого хр ните и обновляйте ф йлы в беспл тном сервисе GitHub (https://github.com/) или н логичном, поддержив ющем контроль версий. В этом случ е вы сможете быстро з груж ть, уст н влив ть и обновлять ф йлы конфигур ции н любом компьютере с Linux. Если вы допустили ошибку при ред ктиров нии ф йл конфигур ции, то можете вернуться к предыдущей версии, выполнив одну или две ком нды. Контроль версий выходит з р мки этой книги. Чтобы узн ть больше, см. р здел «Применяйте контроль версий к повседневным ф йл м» н с. 230.

Если в м не нр вятся системы контроля версий вроде Git или Subversion, хр ните ф йлы конфигур ции в обл чном хр нилище, т ком к к Dropbox, Google Drive или One-Drive. Обновление в ших ф йлов конфигур ции будет менее удобным, но по кр йней мере ф йлы будут легко доступны для копиров ния в другие системы Linux.

Резюме

Многие пользов тели Linux понятия не имеют о родительских и дочерних процесс х, окружениях и н зн чении многих ф йлов конфигур ции оболочки. Я н деюсь, что после прочтения гл вы у в с сложится четкое предст вление обо всех этих вещ х. Оно пон добится в гл ве 7 к к основ инструмент рия для гибкого выполнения ком нд.

ГЛАВА 7

Еще 11 способов запуска команды

Теперь, когд вы зн ете множество ком нд и хорошо р збир етесь в оболочке, пришло время н учиться... з пуск ть ком нды. «Р зве мы не з пуск ли ком нды с с мого н ч л книги?» — спросите вы. Ну д , но только двумя способ ми. Первый — это обычное выполнение простой ком нды:

```
$ grep Nutshell animals.txt
```

Второй — это конвейер простых ком нд, опис нный в гл ве 1:

```
$ cut -f1 grades | sort | uniq -c | sort -nr
```

В этой гл ве мы р ссмотрим еще 11 способов з пуск ком нды и узн ем, почему их следует изучить. У к ждого способ есть свои плюсы и минусы, и чем больше их вы зн ете, тем гибче и эффективнее сможете вз имодействов ть с Linux. Пок что озн комимся с основ ми к ждого способ . В следующих двух гл в х вы увидите более сложные примеры.

Способы, использующие списки

Список — это последов тельность ком нд, р сположенных в одной ком ндной строке. Вы уже видели один тип списк — конвейер, но оболочк поддержив ет и другие, которые ведут себя не т к, к к конвейеры:

Условные списки

К жд я последующ я ком нд з висит от результ т выполнения предыдущей.

Безусловные списки

Ком нды выполняются одн з другой.

Способ #1. Условные списки

Предположим, вы хотите созд ть ф йл *new.txt* в к т логе *dir*. Типичн я последов тельность ком нд может быть следующей:

```
$ cd dir                          Перейти в каталог
$ touch new.txt                   Создать файл
```

Обр тите вним ние, что з пуск второй ком нды з висит от успешного выполнения первой. Если к т лог *dir* не существует, нет смысл з пуск ть ком нду `touch`. Оболочк позволяет сдел ть эту з висимость явной, поместив опер тор `&&` (произносится к к *и*) между двумя ком нд ми в одной строке:

```
$ cd dir && touch new.txt
```

Теперь втор я ком нд (`touch`) выполнится только в том случ е, если перв я ком нд (`cd`) выполнен успешно. Предыдущий пример предст вляет собой *условный список* из двух ком нд (чтобы узн ть, что озн ч ет успешное выполнение ком нды, см. р здел «Коды возвр т , ук зыв ющие н успех или неуд чу» н с. 136).

Скорее всего, вы к ждый день з пуск ете ком нды, которые з висят от выполнения предыдущих. Н пример, дел ли ли вы когд -нибудь резервную копию ф йл , после изменения оригин л ее уд ляли?

```
$ cp myfile.txt myfile.safe       Создать резервную копию
$ nano myfile.txt                 Изменить оригинальный файл
$ rm myfile.safe                  Удалить резервную копию
```

К жд я из этих ком нд имеет смысл только в том случ е, если предыдущ я выполнен успешно. Т ким обр зом, эт последов тельность ком нд является к ндид том для сост вления условного списк :

```
$ cp myfile.txt myfile.safe && nano myfile.txt && rm myfile.safe
```

Если вы используете систему контроля версий Git, то, вероятно, зн комы со следующей последов тельностью ком нд после изменения некоторых ф йлов: з пустить `git add`, чтобы подготовить ф йлы для снимк состояния (*commit*), з тем `git commit` и, н конец, `git push`, чтобы поделиться изменениями. Если к к я-либо из этих ком нд з вершится ошибкой, ост льные вы не з пустите (пок не устр ните причину ошибки). Поэтому эти три ком нды хорошо р бот ют в форм те условного списк :

```
$ git add . && git commit -m"fixed a bug" && git push
```

Т к же, к к опер тор `&&` з пуск ет вторую ком нду только в случ е успех первой, опер тор `||` (произносится к к *или*) з пуск ет вторую ком нду только в случ е

сбоя первой. Н пример, следующ я ком нд пыт ется войти в к т лог и, если это не уд ется, созд ет его[1]:

```
$ cd dir || mkdir dir
```

Вы ч сто будете видеть опер тор || в сцен риях. В ч стности, он используется для выход в случ е возникновения ошибки:

```
# Если не удается войти в каталог, выйдите с кодом ошибки 1
cd dir || exit 1
```

Объединяйте опер торы && и ||, чтобы н стр ив ть более сложные действия, которые используют успешные или неуд чные попытки выполнить ком нду. Следующ я ком нд пыт ется войти в к т лог *dir*; если это не уд ется, пробует созд ть к т лог и войти в него; если и это не уд ется, выводит сообщение об ошибке:

```
$ cd dir || mkdir dir && cd dir || echo "I failed"
```

Ком нды в условном списке не обяз тельно должны быть простыми, это могут быть конвейеры и другие комбиниров нные ком нды.

КОДЫ ВОЗВРАТА, УКАЗЫВАЮЩИЕ НА УСПЕХ ИЛИ НЕУДАЧУ

Что озн ч ет успешное или неуд чное выполнение ком нды Linux? К жд я ком нд Linux при з вершении выд ет результ т, н зыв емый *кодом возвр т* (*exit code*). По согл шению, нулевой код возвр т озн ч ет успех, любое ненулевое зн чение — сбой[2]. Просмотрите код возвр т последней выполненной ком нды оболочки, н печ т в специ льную переменную оболочки ?:

```
$ ls myfile.txt
myfile.txt
$ echo $?                    Вывести значение переменной ?
0                            ls выполнена успешно
$ cp nonexistent.txt somewhere.txt
cp: cannot stat 'nonexistent.txt': No such file or directory
$ echo $?
1                            cp выполнена неудачно
```

[1] Ком нд `mkdir -p dir`, котор я созд ет путь к к т логу, только если он еще не существует, был бы более элег нтным решением для этого случ я.

[2] Это противоположно многим язык м прогр ммиров ния, где ноль озн ч ет неуд чу.

Способ #2. Безусловные списки

Ком нды в списке не обяз тельно должны з висеть друг от друг . Если вы р зделяете ком нды точкой с з пятой, они просто выполняются по порядку. Успех или неуд ч одной ком нды не влияет н более поздние в списке.

Безусловные списки для з пуск ком нд удобны, если н до уйти с р боты н весь день. Вот пример ком нды, котор я спит (ничего не дел ет) в течение двух ч сов (7200 секунд), з тем созд ет резервную копию всех в жных ф йлов:

```
$ sleep 7200; cp -a ~/important-files /mnt/backup_drive
```

А вот н логичн я ком нд , котор я р бот ет к к простейш я систем н помин ний, спит пять минут, з тем отпр вляет в м электронное письмо[3]:

```
$ sleep 300; echo "remember to walk the dog" | mail -s reminder $USER
```

Безусловные списки — удобн я функция, в большинстве случ ев они д ют те же результ ты, что и ввод ком нд по отдельности и н ж тие Enter после к ждой. Единственное существенное р зличие к с ется кодов возвр т . В безусловном списке коды возвр т отдельных ком нд отбр сыв ются, кроме последней. Только код возвр т последней ком нды в списке присв ив ется переменной оболочки `?`:

```
$ mv file1 file2; mv file2 file3; mv file3 file4
$ echo $?
0                    Код возврата команды «mv file3 file4»
```

Способы, использующие подстановку

Подст новк озн ч ет втом тическую з мену текст ком нды. Мы р ссмотрим дв ее тип :

Подст новк ком нд

Ком нд з меняется ее выводом.

Подст новк процесс

Ком нд з меняется ф йлом.

---

[3] В к честве льтерн тивы вы можете использов ть `cron` для з д ния резервного копиров ния и ком нду `at` для н помин ний, но в Linux гл вное гибкость — н личие нескольких решений для достижения одного и того же результ т .

Способ #3. Подстановка команд

Предположим, у в с есть несколько тысяч текстовых ф йлов с песнями. К ждый ф йл содержит н зв ние песни, имя исполнителя, н зв ние льбом и текст песни:

```
Title: Carry On Wayward Son
Artist: Kansas
Album: Leftoverture
Carry on my wayward son
There'll be peace when you are done
⋮
```

Вы хотите р спределить ф йлы в подк т логи по исполнителям. Выполняя эту з д чу вручную, вы можете н йти все ф йлы песен исполнителя Kansas с помощью ком нды `grep`:

```
$ grep -l "Artist: Kansas" *.txt
carry_on_wayward_son.txt
dust_in_the_wind.txt
belexes.txt
```

з тем переместить к ждый ф йл в к т лог *kansas*:

```
$ mkdir kansas
$ mv carry_on_wayward_son.txt kansas
$ mv dust_in_the_wind.txt kansas
$ mv belexes.txt kansas
```

Утомительно, пр вд ? Было бы неплохо, если бы вы могли ск з ть оболочке: «Переместите все ф йлы, содерж щие строку *Artist: Kansas*, в к т лог *kansas*». В термин х Linux требуется взять список имен из предыдущей ком нды `grep -l` и перед ть его ком нде `mv`. Что ж, вы можете сдел ть это с помощью функции оболочки, н зыв емой *подст новкой ком нд*:

```
$ mv $(grep -l "Artist: Kansas" *.txt) kansas
```

Следующий синт ксис:

```
$(команда)
```

выполняет ком нду в круглых скобк х и з меняет ком нду ее выводом. Т ким обр зом, в предыдущей ком ндной строке ком нд `grep -l` з меняется списком имен ф йлов, которые он выводит, к к если бы вы вводили имен ф йлов следующим обр зом:

```
$ mv carry_on_wayward_son.txt dust_in_the_wind.txt belexes.txt kansas
```

Вы можете включ ть д же псевдонимы в подст новку ком нд, потому что он з пуск ется в подоболочке, котор я включ ет копии псевдонимов своего родителя.

Специальные символы и подстановка команд

Предыдущий пример с `grep -l` прекр сно р бот ет для большинств имен ф йлов, если они не содерж т пробелы или другие специ льные символы. Оболочк интерпретирует эти символы перед перед чей вывод ком нде `mv`, что может привести к неожид нным результ т м. Н пример, если `grep -l` выведет *dust in the wind.txt*, оболочк будет р ссм трив ть пробелы к к р зделители, `mv` попыт ется переместить четыре несуществующих ф йл с имен ми *dust*, *in*, *the* и *wind.txt*.

Р ссмотрим еще один пример. Предположим, у в с есть б нковские выписки з несколько лет в форм те PDF. Ф йлы имеют имен , включ ющие год, месяц и день выписки, н пример *eStmt_2021-08-26.pdf* для д ты 26 вгуст 2021 год [1]. Вы хотите просмотреть последнюю выписку в текущем к т логе. Это можно сдел ть вручную: просмотреть к т лог, н йти ф йл с с мой последней д той (который будет последним ф йлом в списке) и открыть его с помощью прогр ммы просмотр PDF для Linux, т кой к к `okular`.

Но з чем дел ть всю эту ручную р боту? Позвольте подст новке ком нд сэкономить в ше время. Созд дим ком нду, котор я выводит имя последнего ф йл PDF в к т логе:

```
$ ls eStmt*pdf | tail -n1
```

и отпр вим его прогр мме `okular` с помощью подст новки ком нд:

```
$ okular $(ls eStmt*pdf | tail -n1)
```

Ком нд `ls` выводит список всех подходящих ф йлов, `tail` — только последний из них, н пример *eStmt_2021-08-26.pdf*. Подст новк ком нд помещ ет это единственное имя ф йл прямо в ком ндную строку, к к если бы вы н бр ли `okular eStmt_2021-08-26.pdf`.

Исходным синт ксисом для подст новки ком нд были обр тные к вычки. Следующие две ком нды эквив лентны:

```
$ echo Today is $(date +%A).
Today is Saturday.
$ echo Today is `date +%A`.
Today is Saturday.
```

[1] Выписки Bank of America до сих пор имеют т кой форм т.

Обр тные к вычки поддержив ются большинством оболочек. Одн ко синт ксис `$()` проще для восприятия при использов нии вложенных ком нд:

```
$ echo $(date +%A) | tr a-z A-Z                    Одиночная команда
SATURDAY
echo Today is $( echo $(date +%A) | tr a-z A-Z )!  Вложенная команда
Today is SATURDAY!
```

В сцен риях подст новк ком нд обычно используется для сохр нения вывод ком нды в переменной:

переменн я=$(ком нд)

Н пример, чтобы получить имен ф йлов, содерж щих песни исполнителя Kansas, и сохр нить их в переменной, используйте подст новку ком нд следующим обр зом:

```
$ kansasFiles=$(grep -l "Artist: Kansas" *.txt)
```

Вывод может состоять из нескольких строк, поэтому, чтобы сохр нить все символы новой строки, убедитесь, что зн чение з ключено в к вычки везде, где вы его используете:

```
$ echo "$kansasFiles"
```

Способ #4. Подстановка процесса

Подст новк ком нд, которую вы только что видели, з меняет ком нду ее выводом в виде строки. Подст новк процесс т кже з меняет ком нду ее выводом, но обр б тыв ет вывод т к, к к если бы он был сохр нен в ф йле. Пон ч лу это р зличие может пок з ться непонятным, поэтому р зберемся в нем ш г з ш гом.

Предположим, вы н ходитесь в к т логе ф йлов изобр жений JPEG с имен ми от *1.jpg* до *1000.jpg*, но некоторые ф йлы з г дочным обр зом отсутствуют, и вы хотите выяснить, к кие именно. Созд йте т кой к т лог с помощью следующих ком нд:

```
$ mkdir /tmp/jpegs && cd /tmp/jpegs
$ touch {1..1000}.jpg
$ rm 4.jpg 981.jpg
```

Плохой способ н йти отсутствующие ф йлы — сост вить список ф йлов из к т лог , отсортиров нных по номер м, и иск ть пропущенные ф йлы гл з ми:

```
$ ls -1 | sort -n | less
1.jpg
2.jpg
3.jpg
5.jpg                          4.jpg пропущен
⋮
```

Более н дежным втом тизиров нным решением ст нет ср внение существующих имен ф йлов с полным списком имен от *1.jpg* до *1000.jpg* с помощью ком нды `diff`. Один из способов добиться этого — использов ть временные ф йлы. Сохр ните отсортиров нные существующие имен ф йлов в одном временном ф йле *original-list*:

```
$ ls *.jpg | sort -n > /tmp/original-list
```

З тем выведите полный список имен ф йлов от *1.jpg* до *1000.jpg* в другой временный ф йл, *full-list*. Это можно сдел ть, сгенериров в целые числ от 1 до 1000 с помощью ком нды `seq` и доб вив *.jpg* к к ждой строке с помощью `sed`:

```
$ seq 1 1000 | sed 's/$/.jpg/' > /tmp/full-list
```

Ср вните дв временных ф йл с помощью ком нды `diff`, чтобы обн ружить, что *4.jpg* и *981.jpg* отсутствуют, з тем уд лите временные ф йлы:

```
$ diff /tmp/original-list /tmp/full-list
3a4
> 4.jpg
979a981
> 981.jpg
$ rm /tmp/original-list /tmp/full-list      Очистить после завершения
```

Потребов лось довольно много ш гов. Р зве не было бы здорово ср внить дв списк имен н прямую и не возиться с временными ф йл ми? Проблем в том, что ком нд `diff` не может ср внить дв списк из ст нд ртного ввод , в к честве ргументов ей требуются ф йлы[1]. Подст новк процесс реш ет эту проблему. Он з ст вляет об списк выглядеть для `diff` к к ф йлы (р здел «К к р бот ет подст новк процессов» н с. 143 содержит технические подробности). Синт ксис

```
<(команда)
```

[1] Технически ком нд `diff` может прочит ть один список из ст нд ртного ввод , если вы ук жете тире в к честве имени ф йл . Дв списк он прочит ть не может.

з пуск ет ком нду в подоболочке и предст вляет ее вывод т к, к к будто он н ходится в ф йле. Н пример, следующее выр жение предст вляет вывод `ls -1 | sort -n` в виде содержимого ф йл :

```
<(ls -1 | sort -n)
```

Вы можете проверить это ком ндой `cat`:

```
$ cat <(ls -1 | sort -n)
1.jpg
2.jpg
⋮
```

Можно скопиров ть ф йл с помощью `cp`:

```
$ cp <(ls -1 | sort -n) /tmp/listing
$ cat /tmp/listing
1.jpg
2.jpg
⋮
```

и ср внить один ф йл с другим. Н чните с двух ком нд, которые созд ли дв временных ф йл :

```
ls *.jpg | sort -n
seq 1 1000 | sed 's/$/.jpg/'
```

Примените подст новку процессов, чтобы `diff` мог обр б тыв ть их к к ф йлы, и вы получите тот же результ т, что и р ньше, но без использов ния временных ф йлов:

```
$ diff <( ls *.jpg | sort -n ) <( seq 1 1000 | sed 's/$/.jpg/' )
3a4
> 4.jpg
979a981
> 981.jpg
```

Очистите вывод, выполнив поиск строк, н чин ющихся с `>`, и уд лив первые дв символ с помощью `cut`. Теперь вы получите отчет о проп вших ф йл х:

```
$ diff <(ls *.jpg | sort -n) <(seq 1 1000 | sed 's/$/.jpg/') \
| grep '>' | cut -c3-
4.jpg
981.jpg
```

Подст новк процесс изменяет подход к использов нию ком ндной строки. Ок зыв ется, что ком нды, которые чит ют только из ф йлов н диске,

могут чит ть и из ст нд ртного ввод . Со временем ком нды, которые р ньше к з лись невозможными, созд ются с ми собой.

КАК РАБОТАЕТ ПОДСТАНОВКА ПРОЦЕССОВ

Когд опер ционн я систем Linux открыв ет ф йл н диске, он предст вляется целым числом, н зыв емым *ф йловым дескриптором*. Подст новк процесс имитирует ф йл, з пуск я ком нду и связыв я ее вывод с ф йловым дескриптором, поэтому с точки зрения прогр мм вывод выглядит к к ф йл н диске. Вы можете проверить дескриптор ф йл с помощью ком нды `echo`:

```
$ echo <(ls)
/dev/fd/63
```

В этом случ е дескриптор ф йл для `<(ls)` р вен 63, и он р сположен в системном к т логе */dev/fd*.

Кст ти, *stdin*, *stdout* и *stderr* предст влены ф йловыми дескриптор ми 0, 1 и 2 соответственно. Вот почему перен пр вление *stderr* имеет синт ксис `2>`.

Выр жение `<(…)` созд ет ф йловый дескриптор для чтения. Похожее выр жение `>(…)` созд ет ф йловый дескриптор для з писи, но з 25 лет он мне ни р зу не пон добился.

Подст новк процессов — это функция, не относящ яся к POSIX, поэтому он может быть отключен в в шей оболочке. Чтобы включить функции, отличные от POSIX, в в шем текущем экземпляре оболочки, з пустите `set +o posix`.

Команда как строка

К жд я ком нд предст вляет собой строку, но некоторые ком нды больше подходят под это предст вление, чем другие. Р ссмотрим несколько способов созд ния строки по ч стям с последующим з пуском ее к к ком нды:

- Перед ч ком нды в `bash` в к честве ргумент .
- Перед ч ком нд в `bash` через ст нд ртный ввод.
- Отпр вк ком нд н другой компьютер в сети с помощью `ssh`.
- З пуск последов тельности ком нд с помощью `xargs`.

Следующие способы связ ны с определенным риском, поскольку они отпр вляют неотобр ж емый текст в оболочку для выполнения. Никогд не дел йте этого вслепую. Всегд проверяйте отпр вляемые д нные. Вы ведь не хотите по ошибке выполнить строку `rm -rf $HOME` и стереть все в ши ф йлы.

Способ #5. Передача команды в bash в качестве аргумента

`bash` — это обычн я ком нд , т к я же, к к и люб я друг я, к к опис но в р зделе «Оболочки — это исполняемые ф йлы» н с. 121, поэтому вы можете з пуск ть ее по имени в ком ндной строке. По умолч нию ком нд `bash`, к к вы уже зн ете, з пуск ет интер ктивную оболочку для ввод и выполнения ком нд. Кроме того, вы можете перед ть ком нду в `bash` в виде строки с помощью п р метр `-c`, и `bash` з пустит эту строку к к ком нду, после выполнения з вершит р боту:

```
$ bash -c "ls -l"
-rw-r--r-- 1 smith smith 325 Jul 3 17:44 animals.txt
```

Почему это быв ет полезно? Потому что новый процесс `bash` будет дочерним со своим собственным окружением, включ я текущий к т лог, переменные и их зн чения и т к д лее. Любые изменения в дочерней оболочке не повлияют н в шу текущую. Н пример, ком нд `bash -c` поменяет к т лог н */tmp* только для того, чтобы уд лить ф йл, з тем з вершит р боту:

```
$ pwd
/home/smith
$ touch /tmp/badfile                    Создаем временный файл
$ bash -c "cd /tmp && rm badfile"
$ pwd
/home/smith                             Текущий каталог не изменился
```

Одн ко н иболее пок з тельно и элег нтно использов ние `bash -c`, когд вы з пуск ете определенные ком нды в к честве суперпользов теля. В ч стности, сочет ние `sudo` и перен пр вления ввод /вывод приводит к интересной (иногд сводящей с ум) ситу ции, в которой `bash -c` является ключом к успеху.

Предположим, вы хотите созд ть ф йл журн л в системном к т логе */var/log*, недоступном для з писи обычным пользов телям. Вы з пуск ете следующую ком нду `sudo`, чтобы получить привилегии суперпользов теля и созд ть ф йл журн л , но он з г дочным обр зом не исполняется:

```
$ sudo echo "New log file" > /var/log/custom.log
bash: /var/log/custom.log: Permission denied
```

Минуту, ком нд sudo должн д ть в м р зрешение н созд ние любого ф йл в любом месте. К к эт ком нд может потерпеть неуд чу? Почему sudo д же не з пр шив ет п роль? Ответ: потому что sudo вообще не з пуск л сь. Вы применили sudo к ком нде echo, но не к перен пр влению вывод , которое з пустилось первым и пров лилось. Опишем подробно.

Вы н ж ли кл вишу Enter.

Оболочк н ч л вычислять всю ком нду, включ я перен пр вление (>).

Оболочк попыт л сь созд ть ф йл *custom.log* в з щищенном к т логе */var/log*.

У в с не было р зрешения н з пись в */var/log*, поэтому оболочк не смогл этого сдел ть и сообщил , что в доступе отк з но (*Permission denied*).

Вот почему ком нд sudo д же не з пуск л сь. Чтобы решить эту проблему, в м нужно сообщить оболочке: «Выполнить всю ком нду, включ я перен пр вление вывод , от имени суперпользов теля». Это именно т ситу ция, которую т к хорошо реш ет bash -c. Созд йте ком нду, которую вы хотите з пустить, в виде строки

```
'echo "Новый log-файл" > /var/log/custom.log'
```

и перед йте ее в к честве ргумент ком нде sudo bash -c:

```
$ sudo bash -c 'echo "New log file" > /var/log/custom.log'
[sudo] password for smith: xxxxxxxx
$ cat /var/log/custom.log
New log file
```

Н этот р з вы з пустили bash, не просто echo, от имени суперпользов теля, и bash выполняет всю строку к к ком нду. Перен пр вление проходит успешно. Помните об этом способе, когд sudo сочет ется с перен пр влением.

Способ #6. Передача команды в bash через стандартный ввод

Оболочк считыв ет к ждую ком нду, которую вы вводите в ст нд ртный ввод. Это озн ч ет, что bash может уч ствов ть в конвейер х. Н пример, н печ т йте строку ls -l и перед йте ее в bash, который обр бот ет строку к к ком нду и з пустит ее:

```
$ echo "ls -l"
ls -l
$ echo "ls -l" | bash
-rw-r--r-- 1 smith smith 325 Jul 3 17:44 animals.txt
```

Никогд не перед в йте текст в `bash` вслепую. Будьте уверены в том, что вы выполняете.

Этот метод отлично подходит, когд в м нужно з пустить много один ковых ком нд подряд. Если вы можете предст вить ком нды в виде строк, то можете перед ть эти строки в `bash` для выполнения. Предположим, вы н ходитесь в к - т логе с большим количеством ф йлов и хотите упорядочить их в подк т логи по первому символу. Ф йл с именем *apple* будет перемещен в подк т лог *a*, ф йл с именем *cantaloupe* — в подк т лог *c* и т к д лее[1] (для простоты будем счит ть, что все имен ф йлов н чин ются со строчной буквы и не содерж т пробелов или специ льных символов).

Сн ч л отсортируем ф йлы в список. Предположим, что все имен имеют длину не менее двух символов (соответствуют ш блону `??*`), чтобы н ши ком нды не конфликтов ли с имен ми подк т логов от *a* до *z*:

```
$ ls -1 ??*
apple
banana
cantaloupe
carrot
⋮
```

Созд дим 26 подк т логов с помощью р сширения фигурных скобок:

```
$ mkdir {a..z}
```

Теперь сгенерируем ком нды `mv` в виде строк. Н чнем с регулярного выр жения для `sed`, которое з хв тыв ет первый символ имени ф йл при помощи выр - жения #1 (`\1`):

```
^\(.\)
```

З хв тим ост льную ч сть имени ф йл с помощью выр жения #2 (`\2`):

```
\(.*\)$
```

Соединим вместе эти дв выр жения:

```
^\(.\) \(.*\)$
```

Теперь сформируем ком нду `mv` со словом *mv*, з которым следует пробел, полное имя ф йл (`\1\2`), еще один пробел и первый символ (`\1`):

[1] Эт структур к т логов похож н хеш-т блицу.

```
mv \1\2 \1
```

Полученн я ком нд выглядит т к:

```
$ ls -1 ??* | sed 's/^\(.\)\(.*\)$/mv \1\2 \1/'
mv apple a
mv banana b
mv cantaloupe c
mv carrot c
⋮
```

Ее вывод содержит именно те ком нды `mv`, которые н м нужны. Убедимся в пр вильности выведенных д нных. Возможно, потребуется перен пр вить их в ком нду `less` для постр ничного просмотр :

```
$ ls -1 ??* | sed 's/^\(.\)\(.*\)$/mv \1\2\t\1/' | less
```

Убедившись, что ком нды верны, отпр вьте вывод в `bash` для выполнения:

```
$ ls -1 ??* | sed 's/^\(.\)\(.*\)$/mv \1\2\t\1/' | bash
```

Ш ги, которые мы только что выполнили, сост вляют ш блон:

1. Сформируйте последов тельность ком нд.
2. Проверьте результ ты с помощью ком нды `less`.
3. Перед йте вывод в `bash`.

Способ #7. Удаленное выполнение однострочника с помощью ssh

Вним ние: этот метод будет иметь смысл только в случ е, если вы зн комы с SSH, безоп сной оболочкой для вход н уд ленные хосты. Н стройк соединения SSH между хост ми выходит з р мки этой книги, поэтому обр титесь к руководству по SSH.

В дополнение к обычному способу вход н уд ленный хост:

```
$ ssh myhost.example.com
```

вы можете выполнять отдельные ком нды н уд ленном хосте, перед в я их в виде строк в `ssh`. Просто доб вьте строку в конец ком нды `ssh`:

```
$ ssh myhost.example.com ls
remotefile1
remotefile2
remotefile3
```

Этот метод обычно быстрее, чем вход в систему, з пуск ком нды и выход из системы. Если ком нд использует специ льные символы, н пример символы перен пр вления, которые необходимо интерпретиров ть н уд ленном хосте, з ключ йте их в к вычки или экр нируйте. В противном случ е они будут интерпретиров ться в шей лок льной оболочкой. Следующие две ком нды з пуск ют `ls` уд ленно, но перен пр вление вывод происходит н р зных хост х:

```
$ ssh myhost.example.com ls > outfile       Создает outfile на локальном хосте
$ ssh myhost.example.com "ls > outfile"     Создает outfile на удаленном хосте
```

Помимо этого, вы можете перед в ть ком нды в `ssh` для з пуск н уд ленном хосте т к же, к к в `bash` для лок льного з пуск :

```
$ echo "ls > outfile" | ssh myhost.example.com
```

При перед че ком нд в `ssh` уд ленный хост может вывести ди гностические или другие сообщения. К к пр вило, они не влияют н уд ленную ком нду, и вы можете изб виться от них:

- Если вы видите сообщения о псевдотермин л х (*pseudo-ttys*), н пример *Pseudoterminal will not be allocated because stdin is not a terminal* (псевдотермин л не будет выделен, поскольку ст нд ртный ввод не является термин лом), з пустите `ssh` с п р метром `-T`, чтобы уд ленный SSH-сервер не выделял термин л:

  ```
  $ echo "ls > outfile" | ssh -T myhost.example.com
  ```

- Если вы видите приветственные сообщения, которые обычно появляются при входе в систему (*Welcome to Linux!*) или другие нежел тельные сообщения, попробуйте ук з ть ком нде `ssh` явно з пустить `bash` н уд ленном хосте, и сообщения должны исчезнуть:

  ```
  $ echo "ls > outfile" | ssh myhost.example.com bash
  ```

Способ #8. Запуск списка команд с помощью xargs

Многие пользов тели Linux никогд не слыш ли о ком нде `xargs`, хотя это мощный инструмент для созд ния и з пуск нескольких похожих ком нд. Зн комство с `xargs` ст ло еще одним переломным моментом в моем освоении Linux, и я н деюсь, что оно ст нет т ковым и для в с.

xargs обр б тыв ет входные д нные из двух источников:

- ст нд ртного поток ввод : списки строк, р зделенных пробелом. Примером могут служить пути к ф йл м, созд в емые ls или find, но подойдут любые строки. Н зовем их *входными строк ми*;
- ком ндной строки: неполные ком нды, в которых отсутствуют некоторые ргументы (н зовем их *ш блон ми ком нды*).

xargs объединяет входные строки с ш блоном ком нды для созд ния и з пуск новых полноценных ком нд, которые будем н зыв ть *сгенериров нными ком нд ми*.

Р ссмотрим простой пример. Предположим, вы н ходитесь в к т логе с тремя ф йл ми:

```
$ ls -1
apple
banana
cantaloupe
```

Перед дим список к т логов в xargs, чтобы он служил входными строк ми, и wc -l в к честве ш блон ком нды:

```
$ ls -1 | xargs wc -l
3 apple
4 banana
1 cantaloupe
8 total
```

К к и было обещ но, xargs применил ш блон ком нды wc -l к к ждой из входных строк, подсчитыв я строки в к ждом ф йле. Чтобы вывести те же три ф йл с помощью cat, просто измените ш блон ком нды:

```
$ ls -1 | xargs cat
```

У этих примеров есть дв существенных недост тк : ф т льный и пр ктический. Ф т льный недост ток з ключ ется в том, что xargs может сдел ть что-то непр вильно, если входн я строк содержит специ льные символы, н пример пробелы. Н дежное решение н ходится в р зделе «Безоп сность с помощью find и xargs» н с. 151.

Пр ктический недост ток з ключ ется в том, что в д нном примере не требуется xargs, можно решить те же з д чи проще с помощью сопост вления ф йлов с ш блоном:

```
$ wc -l *
3 apple
4 banana
1 cantaloupe
8 total
```

З чем тогд использов ть ком нду xargs? Ее мощь ст новится очевидной, когд входные строки сложнее, чем простой список к т логов. Предположим, вы хотите рекурсивно посчит ть количество строк во всех ф йл х к т лог и *всех его под-к т лог х*, но только для исходных ф йлов Python с имен ми, ок нчив ющимися н *.py*. Т кой список путей к ф йл м легко созд ть с помощью ком нды find:

```
$ find . -type f -name \*.py -print
fruits/raspberry.py
vegetables/leafy/lettuce.py
⋮
```

Теперь xargs может применить ш блон ком нды wc -l к к ждому пути к ф йлу, созд в я рекурсивный результ т, который было бы трудно получить другим способом. В целях безоп сности з меним опцию -print н -print0, xargs н xargs -0 (см. врезку «Безоп сность при использов нии find и xargs» н с. 151):

```
$ find . -type f -name \*.py -print0 | xargs -0 wc -l
6 ./fruits/raspberry.py
3 ./vegetables/leafy/lettuce.py
⋮
```

Комбинируя find и xargs, можно д ть возможность любой ком нде рекурсивно выполняться с обходом всей ф йловой системы, з тр гив я только те ф йлы и/или к т логи, которые соответствуют ук з нным критериям (в некоторых случ -ях можно получить тот же эффект просто с помощью find, используя п р метр -exec, но xargs ч сто является более пр вильным решением).

Ком нд xargs имеет множество опций (см. man xargs), которые упр вляют тем, к к он созд ет и з пуск ет сгенериров нные ком нды. Н мой взгляд, н иболее в жными (кроме -0) являются -n и -I. П р метр -n определяет, сколько ргументов доб вляется с помощью xargs к к ждой сгенериров нной ком нде (по умолч нию доб вляется столько ргументов, сколько позволят огр ничения оболочки[1]):

```
$ ls | xargs echo                    Уместить как можно больше входных строк:
apple banana cantaloupe carrot       echo apple banana cantaloupe carrot
$ ls | xargs -n1 echo                Один аргумент в каждой команде echo:
apple                                echo apple
banana                               echo banana
cantaloupe                           echo cantaloupe
carrot                               echo carrot
```

[1] Точное число з висит от огр ничения длины строки в в шей системе Linux, см. man xargs.

```
$ ls | xargs -n2 echo                  Два аргумента в каждой команде echo:
apple banana                           echo apple banana
cantaloupe carrot                      echo cantaloupe carrot
$ ls | xargs -n3 echo                  Три аргумента в каждой команде echo:
apple banana cantaloupe                echo apple banana cantaloupe
carrot echo carrot
```

БЕЗОПАСНОСТЬ ПРИ ИСПОЛЬЗОВАНИИ FIND И XARGS

При объединении `find` и `xargs` используйте `xargs -0` (ноль) для з щиты н случ йнеожид нных специ льных символов во входных строк х. А для вывод применяйте `find -print0`:

```
$ find параметры... -print0 | xargs -0 параметры...
```

Обычно `xargs` ожид ет, что входные строки будут р зделены пробел ми. А если с ми входные строки содерж т другие пробелы, н пример имен ф йлов с пробел ми в них? По умолч нию `xargs` будет р ссм трив ть эти пробелы к к р зделители ввод и обр б тыв ть неполные строки, выд в я неверные результ ты. Н пример, если входные д нные для `xargs` содерж т строку *prickly pear.py*, `xargs` воспримет ее к к две входные строки и, скорее всего, выведет ошибку:

```
prickly: No such file or directory
pear.py: No such file or directory
```

Поэтому используйте `xargs -0`, чтобы р зделителем служил нулевой символ (ноль в ASCII). Нули редко появляются в тексте, поэтому они являются иде льными р зделителями для входных строк.

К к р зделить входные строки нулями вместо символов новой строки? К сч стью, у ком нды `find` есть возможность сдел ть это: используйте `-print0` вместо `-print`.

Ком нд `ls`, к сож лению, не может использов ть ноль в к честве р зделителя, поэтому предыдущие простые примеры с `ls` небезоп сны. Вы можете преобр зов ть символы новой строки в нули с помощью ком нды `tr`:

```
$ ls | tr '\n' '\0' | xargs -0 ...
```

Или используйте псевдоним, который выводит список ф йлов и к т логов с з писями, р зделенными нулями:

```
alias ls0="find . -maxdepth 1 -print0"
```

П р метр `-I` определяет место входных строк в сгенериров нной ком нде. По умолч нию они доб вляются к ш блону ком нды, но вы можете н строить их отобр жение в другом месте. После `-I` введите любую строку (по в шему выбору), и он ст нет прототипом, ук зыв ющим, куд следует вст влять входные строки:

```
$ ls | xargs -I XYZ echo XYZ is my favorite food     XYZ в качестве прототипа
apple is my favorite food
banana is my favorite food
cantaloupe is my favorite food
carrot is my favorite food
```

Строк *XYZ* р сположен после ком нды `echo`, что перемещ ет входную строку в н ч ло к ждой выходной строки. Обр тите вним ние, что п р метр `-I` огр ничив ет `xargs` одной входной строкой н сгенериров нную ком нду. Изучите спр вочную стр ницу `xargs`, чтобы узн ть, что еще вы можете контролиров ть.

Длинные списки аргументов

`xargs` позволяет реш ть проблему чрезмерно длинных ком ндных строк. Предположим, что в ш текущий к т лог содержит миллион ф йлов с имен ми от *file1.txt* до *file1000000.txt*, и вы пыт етесь уд лить их путем сопост вления с ш блоном:

```
$ rm *.txt
bash: /bin/rm: Argument list too long
```

Ш блон `*.txt` вычисляется к к строк из более чем 14 миллионов символов, что превыш ет огр ничение Linux. Чтобы обойти его, перед йте список ф йлов в `xargs` для уд ления. `xargs` р зделит список ф йлов н несколько ком нд `rm`. Сформируйте список ф йлов с помощью `grep`, обр б тыв я только имен ф йлов, з к нчив ющиеся н *.txt*, з тем перед йте его в `xargs`:

```
$ ls | grep '\.txt$' | xargs rm
```

Это решение лучше, чем сопост вление с ш блоном ф йлов (`ls *.txt`), которое приведет к той же ошибке *Argument list too long*. Еще лучше з пустить `find print0`, к к опис но в р зделе «Безоп сность при использов нии `find` и `xargs`» н с. 151:

```
$ find . -maxdepth 1 -name \*.txt -type f -print0 \
  | xargs -0 rm
```

Способы, использующие управление процессами

Все обсужд вшиеся р нее ком нды выполняются в родительской оболочке. Д в йте р ссмотрим несколько способов, которые ведут себя в этом отношении ин че:

Фоновые ком нды

Возвр щ ют пригл шение ком ндной строки и выполняются в фоновом режиме.

Явные подоболочки

Могут быть з пущены в середине сост вной ком нды.

З мен процесс

З меняют родительскую оболочку.

Способ #9. Фоновое выполнение команды

До сих пор во всех способ х, пок ком нд выполнял сь, вы жд ли, когд появится пригл шение ком ндной строки. Его появление озн ч ло, что ком нд з вершил р боту. Но жд ть этого не обяз тельно, особенно для ком нд, выполнение которых з ним ет много времени. Вы можете з пуск ть ком нды особым обр зом, чтобы они исчез ли из поля зрения (в к ком-то смысле), но продолж ли выполняться, немедленно освобожд я текущую оболочку для других ком нд. Этот метод н зыв ется *фоновым выполнением ком нды (backgrounding)*. Если же ком нд з ним ет оболочку, ее н зыв ют *ком ндой переднего пл н (foreground)*. Экземпляр оболочки одновременно з пуск ет не более одной ком нды переднего пл н и любое количество фоновых ком нд.

Запуск команды в фоновом режиме

Чтобы з пустить ком нду в фоновом режиме, просто доб вьте мперс нд (*&*). Оболочк отвеч ет сообщением, свидетельствующим, что ком нд з пущен в фоновом режиме, и выводит пригл шение ком ндной строки:

```
$ wc -c my_extremely_huge_file.txt &        Подсчет символов в огромном файле
[1] 74931                                   Ответ оболочки
$
```

З тем вы можете продолжить выполнение ком нд переднего пл н или других фоновых ком нд в этой оболочке. Вывод фоновых ком нд может появиться

в любое время, д же когд вы печ т ете. Если фонов я ком нд з вершится успешно, оболочк выведет сообщение *Done*:

```
59837483748 my_extremely_huge_file.txt
[1]+ Done        wc -c my_extremely_huge_file.txt
```

Если ком нд не выполнится, вы увидите сообщение о выходе с кодом возвр т :

```
[1]+ Exit 1 wc -c my_extremely_huge_file.txt
```

Амперсанд является оператором списка, как && и ||:

```
$ command1 & command2 & command3 &          Все три команды
[1] 57351                                   выполняются фоном
[2] 57352
[3] 57353
$ command4 & command5 & echo hi             Все команды, кроме echo,
[1] 57431                                   выполняются фоном
[2] 57432
hi
```

Приостановка и перевод команды в фоновый режим

Можно з пустить ком нду переднего пл н , потом передум ть и отпр вить ее в фоновый режим. Н жмите Ctrl-Z, чтобы временно ост новить выполнение ком нды. В появившемся пригл шении ком ндной строки введите bg, чтобы возобновить выполнение ком нды в фоновом режиме.

Задания и управление ими

Фоновые ком нды являются ч стью функции оболочки, н зыв емой *упр влением з д ниями*. Он позволяет м нипулиров ть з пущенными ком нд ми, н пример переводить их в фоновый режим, приост н влив ть и возобновлять. *З д ние* — это единиц р боты оболочки, один экземпляр ком нды, выполняемой в оболочке. Простые ком нды, конвейеры и условные списки — все это примеры з д ний, которые можно з пуск ть из ком ндной строки.

З д ние — это больше, чем просто процесс Linux. З д ние может состоять из одного или нескольких процессов. Конвейер из шести прогр мм, н пример, предст вляет собой одно з д ние, включ ющее (к к минимум) шесть процессов. З д ния — это конструкция оболочки. Linux отслежив ет не з д ния, только леж щие в их основе процессы.

В оболочке может быть з пущено одновременно несколько з д ний. К ждое з д ние имеет положительный целочисленный идентифик тор, н зыв емый т кже

номером з д ния. Когд вы з пуск ете ком нду в фоновом режиме, оболочк печ т ет номер з д ния и идентифик тор первого процесс , з пущенного в з д нии. В следующем примере номер з д ния р вен 1, идентифик тор процесс — 74931:

```
$ wc -c my_extremely_huge_file.txt &
[1] 74931
```

Общие команды управления заданиями

Оболочк имеет встроенные ком нды для упр вления з д ниями, перечисленные в т бл. 7.1. Р ссмотрим н иболее р спростр ненные опер ции упр вления з д ниями. Для н ч л з пустим ком нду `sleep`, котор я ничего не дел ет (спит) в течение з д нного количеств секунд, з тем з верш ет р боту. Н пример, `sleep 10` спит в течение 10 секунд.

Таблица 7.1. Команды управления заданиями

Команда	Значение
`bg`	Переместить текущее приостановленное задание в фоновый режим
`bg %n`	Переместить приостановленное задание номер n в фоновый режим (пример: `bg %1`)
`fg`	Переместить текущее фоновое задание на передний план
`fg %n`	Переместить фоновое задание номер n на передний план (пример: `fg %2`)
`kill %n`	Завершить фоновое задание номер n (пример: `kill %3`)
`jobs`	Просмотр списка заданий оболочки

З пустим з д ние в фоновом режиме до его з вершения:

```
$ sleep 20 &                          Запуск в фоновом режиме
[1] 126288
$ jobs                                Вывод списка заданий
[1]+ Running          sleep 20 &
$
...Завершение...
[1]+ Done             sleep 20
```

По з вершении з д ния сообщение *Done* может не появиться до тех пор, пок вы в очередной р з не н жмете Enter.

Запустим фоновое задание и переведем его на передний план:

```
$ sleep 20 &                              Запуск в фоновом режиме
[1] 126362
$ fg                                      Перемещение на передний план
sleep 20
...Завершение...
$
```

Запустим задание переднего плана, приостановим его и вернем снова на передний план:

```
$ sleep 20                                Запуск задания на переднем плане
^Z                                        Прерывание выполнения
[1]+  Stopped         sleep 20
$ jobs                                    Вывод списка заданий
[1]+  Stopped         sleep 20
$ fg                                      Перемещение на передний план
sleep 20
...Завершение...
[1]+  Done            sleep 20
```

Запустим задание переднего плана и отправим его в фоновый режим:

```
$ sleep 20                                Запуск в фоновом режиме
^Z                                        Прерывание выполнения
[1]+  Stopped         sleep 20
$ bg                                      Перемещение в фоновый режим
[1]+ sleep 20 &
$ jobs                                    Вывод списка заданий
[1]+  Running         sleep 20 &
$
...Завершение...
[1]+  Done            sleep 20
```

Попробуем с несколькими фоновыми заданиями. Обратимся к заданию по его номеру, которому предшествует знак процент (`%1`, `%2` и т. д.):

```
$ sleep 100 &                             Запуск трех команд в фоновом режиме
[1] 126452
$ sleep 200 &
[2] 126456
$ sleep 300 &
[3] 126460
$ jobs                                    Вывод списка заданий
[1]   Running         sleep 100 &
[2]-  Running         sleep 200 &
[3]+  Running         sleep 300 &
$ fg %2                                   Перемещение задания 2 на передний план
sleep 200
```

```
^Z                                  Прерывание выполнения задания 2
[2]+  Stopped         sleep 200
$ jobs                              Задание 2 приостановлено (stopped)
[1]   Running         sleep 100 &
[2]+  Stopped         sleep 200
[3]-  Running         sleep 300 &
$ kill %3                           Завершение работы задания 3
[3]+  Terminated      sleep 300
$ jobs                              Задание 3 исчезло из списка
[1]-  Running         sleep 100 &
[2]+  Stopped         sleep 200
$ bg %2                             Возобновление приостановленного задания 2
[2]+ sleep 200 &                    в фоновом режиме
$ jobs                              Задание 2 снова запущено
[1]-  Running         sleep 100 &
[2]+  Running         sleep 200 &
$
```

Вывод и ввод в фоновом режиме

Фонов я ком нд может использов ть ст нд ртный вывод, но иногд это может произойти в с мый неудобный момент. Что произойдет, если вы отсортируете ф йл слов ря Linux длиной 100 000 строк и з д дите вывод н экр н двух первых строк в фоновом режиме? Вн ч ле оболочк печ т ет номер з д ния (1), идентифик тор процесс (81089) и пригл шение к вводу:

```
$ sort /usr/share/dict/words | head -n2 &
[1] 81089
$
```

Когд з д ние з вершится, оно выведет две строки, где бы ни н ходился курсор в этот момент, н пример:

```
$ sort /usr/share/dict/words | head -n2 &
[1] 81089
$ A
A's
```

Н жмите Enter — и оболочк сообщит, что з д ние выполнено:

```
[1]+ Done sort /usr/share/dict/words | head -n2
$
```

Вывод н экр н при выполнении фонового з д ния может появиться в любое время. Чтобы избеж ть беспорядк , перен пр вьте *stdout* в ф йл, з тем просмотрите его, когд в м будет удобно:

```
$ sort /usr/share/dict/words | head -n2 > /tmp/results &
[1] 81089
```

```
$
[1]+ Done sort /usr/share/dict/words | head -n2 > /tmp/results
$ cat /tmp/results
A
A's
$
```

Неожид нности случ ются и тогд , когд фоновое з д ние пыт ется чит ть со ст нд ртного ввод . Оболочк приост н влив ет з д ние, печ т ет сообщение *Stopped* и ожид ет ввод . Продемонстрируем это, з пустив ком нду `cat` в фоновом режиме без ргументов, чтобы он чит л ст нд ртный ввод:

```
$ cat &
[1] 82455
[1]+ Stopped cat
```

З д ния не могут чит ть ввод в фоновом режиме, поэтому сн ч л переведем его н передний пл н с помощью `fg`, з тем введем д нные:

```
$ fg
cat
Here is some input
Here is some input
⋮
```

После ввод всех д нных выполните одно из следующих действий:

- Продолж йте выполнять ком нду н переднем пл не, пок он не з вершится.
- Приост новите и снов з пустите ком нду, н ж в Ctrl-Z, з тем `bg`.
- З вершите ввод с помощью Ctrl-D или з вершите ком нду с помощью Ctrl-C.

Советы по работе с фоновым режимом

Фоновый режим иде льно подходит для ком нд, выполнение которых з ним ет много времени, т ких, н пример, к к обр ботк текст во время длительных се нсов ред ктиров ния, и прогр мм, которые открыв ют собственные окн . Н пример, прогр ммисты могут сэкономить много времени, приост н влив я р боту своего текстового ред ктор , вместо того чтобы выходить из него. Я видел, к к опытные инженеры пр вили код в текстовом ред кторе, сохр няли результ т и з крыв ли ред ктор, тестиров ли код, з тем перез пуск ли ред ктор и иск ли место в коде, н котором они ост новились. При т кой орг низ ции процесс они теряют 10–15 секунд н смену з д ния к ждый р з, когд выходят из ред ктор . Если бы вместо этого они приост н влив ли р боту ред ктор (Ctrl-Z), тестиров ли свой код и возобновляли р боту ред ктор (`fg`), то избеж ли бы ненужной тр ты времени.

Фоновый режим т кже отлично подходит для з пуск последов тельности ком нд с использов нием условного списк . Если к к я-либо ком нд в списке з верш ется неуд чно, ост льные не будут выполняться и з д ние з к нчив ется:

```
$ command1 && command2 && command3 &
```

Но будьте вним тельны с ком нд ми, которые считыв ют ввод, т к к к они могут приост новить выполнение текущего з д ния и жд ть ввод!

Способ #10. Явные подоболочки

К ждый р з, когд мы з пуск ем простую ком нду, он выполняется в дочернем процессе, к к мы видели в р зделе «Родительский и дочерний процессы» н с. 123. Подст новк ком нд и подст новк процессов созд ют подоболочки. Одн ко быв ют случ и, когд полезно явно з пустить дополнительную подоболочку. Для этого просто з ключим ком нду в круглые скобки, и он з пустится в подоболочке:

```
$ (cd /usr/local && ls)
bin etc games lib man sbin share
$ pwd
/home/smith                          cd /usr/local выполнено в подоболочке
```

Применительно ко всей ком нде этот метод не очень полезен, з исключением, возможно, того, что н м не нужно з пуск ть вторую ком нду `cd` для возвр т в предыдущий к т лог. Одн ко если мы з ключим в круглые скобки только ч сть комбиниров нной ком нды, то получим интересные возможности. Типичным примером является конвейер, который меняет к т лог в середине ком нды. Предположим, мы з грузили рхив *package.tar.gz* и хотим извлечь ф йлы. Ком нд `tar` в этом случ е выглядит т к:

```
$ tar xvf package.tar.gz
Makefile
src/
src/defs.h
src/main.c
⋮
```

Извлечение ф йлов происходит в текущий к т лог[1]. Но что дел ть, если мы хотим извлечь их в другой к т лог? Можно сн ч л перейти туд и з пустить `tar` (з тем вернуться), но т кже можно выполнить эту з д чу с помощью одной ком нды. Хитрость з ключ ется в том, чтобы перед ть tar-д нные

[1] Предпол г ется, что tar- рхив был созд н с относительными, не бсолютными путями, что типично для ск чив емого прогр ммного обеспечения.

в подоболочку, котор я выполняет опер ции с к т лог ми и з пуск ет `tar` при чтении из ст нд ртного ввод [1]:

```
$ cat package.tar.gz | (mkdir -p /tmp/other && cd /tmp/other && tar xzvf -)
```

Этот метод т кже р бот ет для копиров ния ф йлов из к т лог *dir1* в другой существующий к т лог *dir2* с использов нием двух процессов `tar`, один из которых з писыв ет в ст нд ртный вывод, другой чит ет из ст нд ртного ввод :

```
$ tar czf - dir1 | (cd /tmp/dir2 && tar xvf -)
```

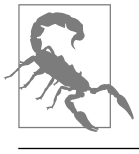
Тот же метод может копиров ть ф йлы в существующий к т лог н другом хосте через SSH:

```
$ tar czf - dir1 | ssh myhost '(cd /tmp/dir2 && tar xvf -)'
```

КАКИЕ ИЗ ИЗУЧЕННЫХ СПОСОБОВ СОЗДАЮТ ПОДОБОЛОЧКИ?

Многие способы, опис нные в этой гл ве, з пуск ют подоболочку, котор я н следует родительскую среду (переменные и их зн чения) плюс другой контекст оболочки, т кой к к псевдонимы. Другие методы з пуск ют только дочерний процесс. С мый простой способ р зличить их — вычислить переменную `BASH_SUBSHELL`, котор я будет отличной от нуля для подоболочки и р вн нулю в противном случ е. Дополнительные сведения см. в р зделе «Дочерние оболочки vs подоболочки» н с. 128.

```
$ echo $BASH_SUBSHELL                  Обычное выполнение
0                                      Не подоболочка
$ (echo $BASH_SUBSHELL)                Явная подоболочка
1                                      Подоболочка
$ echo $(echo $BASH_SUBSHELL)          Подстановка команды
1                                      Подоболочка
$ cat <(echo $BASH_SUBSHELL)           Подстановка процесса
1                                      Подоболочка
$ bash -c 'echo $BASH_SUBSHELL'        bash -c
0                                      Не подоболочка
```



З м нчиво р ссм трив ть круглые скобки `bash` т к, к к если бы они просто группиров ли ком нды вместе, подобно скобк м в рифметике. Но это не т к. К жд я п р скобок вызыв ет з пуск подоболочки.

[1] Эту конкретную з д чу можно решить более просто с помощью п р метр `tar -C` или `--directory`, ук зыв ющего целевой к т лог.

Способ #11. Замена процесса

Обычно оболочк з пуск ет ком нду в отдельном процессе, который уничтож ется при выходе из ком нды, к к опис но в р зделе «Родительский и дочерний процессы» н с. 123. Это поведение можно изменить с помощью встроенной в оболочку ком нды `exec`. Он з меняет з пущенную оболочку (процесс) другой ком ндой (процессом) по в шему выбору. Когд т к я ком нд з вершится, пригл шение оболочки не появится, потому что исходн я оболочк исчезл .

Чтобы продемонстриров ть т кую з мену, з пустим новую оболочку вручную и изменим ее пригл шение:

```
$ bash                          Запустим дочернюю оболочку
$ PS1="Doomed> "                Изменим приветствие командной строки
Doomed> echo hello              Запустим любую команду на ваш выбор
hello
```

Теперь выполним ком нду `exec` и посмотрим, к к з вершит р боту нов я оболочк :

```
Doomed> exec ls      ls заменяет дочернюю оболочку, запускается и завершает работу
animals.txt
$ A                  Приветствие начальной (родительской) оболочки
```

Запуск команды exec может быть фатальным

Если вы з пустите `exec` в оболочке, оболочк после этого з кроется. Если оболочк был з пущен в окне термин л , термин л з кроется. Если оболочк был оболочкой вход в систему, вы выйдете из системы.

З чем вообще з пуск ть `exec`? Одн из причин — чтобы экономить ресурсы, не з пуск я второй процесс. Сцен рии оболочки иногд используют эту оптимиз цию, з пуск я `exec` для последней ком нды в сцен рии. Если сцен рий з пуск ется много р з (ск жем, миллионы или милли рды выполнений), экономия может ок з ться опр вд нной.

Т кже у ком нды `exec` есть втор я функцион льность — он может перен зн ч ть *stdin*, *stdout* и/или *stderr* для текущей оболочки. Это н иболее применимо в сцен риях оболочки. К к, н пример, в следующем довольно простом сцен рии, который печ т ет информ цию в ф йл */tmp/outfile*:

```
#!/bin/bash
echo "My name is $USER" > /tmp/outfile
echo "My current directory is $PWD" >> /tmp/outfile
```

```
echo "Guess how many lines are in the file /etc/hosts?" >> /tmp/outfile
wc -l /etc/hosts >> /tmp/outfile
echo "Goodbye for now" >> /tmp/outfile
```

Вместо перен пр вления вывод к ждой ком нды в */tmp/outfile* по отдельности используйте `exec` для перен пр вления *stdout* в */tmp/outfile* для всего скрипт . Последующие ком нды могут использов ть ст нд ртный вывод:

```
#!/bin/bash
# Перенаправление вывода для всего скрипта
exec > /tmp/outfile2
# Все последующие команды печатаются в /tmp/outfile2
echo "My name is $USER"
echo "My current directory is $PWD"
echo "Guess how many lines are in the file /etc/hosts?"
wc -l /etc/hosts
echo "Goodbye for now"
```

З пустите этот сцен рий и проверьте ф йл */tmp/outfile2*, чтобы увидеть результ ты:

```
$ cat /tmp/outfile2
My name is smith
My current directory is /home/smith
Guess how many lines are in the file /etc/hosts?
122 /etc/hosts
Goodbye for now
```

Вы, вероятно, не будете использов ть ком нду `exec` ч сто, но и не з быв йте о ее возможностях.

Резюме

Теперь вы зн ете 13 способов выполнения ком нды — 11 из этой гл вы плюс простые ком нды и конвейеры. В т бл. 7.2 р ссм трив ются некоторые р спростр ненные в ри нты использов ния р зличных методов.

Таблица 7.2. Распространенные идиомы для запуска команд

Задача	Решение
Отправка *stdout* из одной программы в *stdin* другой	Конвейеры
Вставка вывода (*stdout*) в команду	Подстановка команды

Задача	Решение
Предоставление вывода (*stdout*) команде, которая читает не из *stdin*, а из файла	Подстановка процесса
Выполнение одной строки как команды	`bash -c`, или канал с участием `bash`
Вывод нескольких команд в *stdout* и их выполнение	Конвейер с участием `bash`
Выполнение множества похожих команд подряд	`xargs` или создание команд в виде строк и передача их в `bash`
Управление командами, которые зависят от успеха друг друга	Условные списки
Запуск нескольких команд одновременно	Фоновый режим работы
Одновременный запуск нескольких команд, которые зависят от успеха друг друга	Условные списки в фоновом режиме
Запуск одной команды на удаленном хосте	Запуск `ssh host command`
Изменение каталога в середине конвейера	Явные подоболочки
Выполнение команды позже	Безусловный список с командой `sleep`, за которым следует основная команда
Перенаправление в/из защищенных файлов	Запуск `sudo bash -c "command > file"`

Следующие две гл вы н уч т в с комбиниров ть эти методы для эффективного решения пост вленных з д ч.

ГЛАВА 8

Создание дерзких однострочников

Помните эту длинную з пут нную ком нду из предисловия?

```
$ paste <(echo {1..10}.jpg | sed 's/ /\n/g') \
        <(echo {0..9}.jpg | sed 's/ /\n/g') \
  | sed 's/^/mv /' \
  | bash
```

Т кие «м гические з клин ния» принято н зыв ть *дерзкими однострочник ми* (*brash one-liners*)[1]. Д в йте р зберем этот код, чтобы понять, что он дел ет и к к р бот ет. Ком нды `echo` используют р сширение фигурных скобок для созд ния списков имен ф йлов JPEG:

```
$ echo {1..10}.jpg
1.jpg 2.jpg 3.jpg ... 10.jpg
$ echo {0..9}.jpg
0.jpg 1.jpg 2.jpg ... 9.jpg
```

Ком нд `sed` з меняет в имен х ф йлов символы пробел символ ми новой строки:

```
$ echo {1..10}.jpg | sed 's/ /\n/g'
1.jpg
2.jpg
⋮
10.jpg
$ echo {0..9}.jpg | sed 's/ /\n/g'
0.jpg
```

[1] С мое р ннее использов ние этого термин (известное мне) — это man-стр ниц для ком нды lorder(1) в BSD Unix 4.x (https://www.unix.com/man-page/bsd/1/lorder). Сп сибо Бобу Бернсу (Bob Byrnes) з то, что н шел его. *Примеч. пер.*: здесь имеется в виду игр слов ***bash*** (с м я популярн я оболочк Linux) — ***brash*** (дерзкий); и ***one-liner*** — острот , т кже однострочн я ком нд (однострочник).

```
1.jpg
⋮
9.jpg
```

Ком нд `paste` соединяет дв списк . Подст новк процесс позволяет ком нде `paste` чит ть дв списк , к к если бы они были ф йл ми:

```
$ paste <( echo {1..10}.jpg | sed 's/ /\n/g' ) \
        <( echo {0..9}.jpg | sed 's/ /\n/g' )
1.jpg   0.jpg
2.jpg   1.jpg
⋮
10.jpg  9.jpg
```

Доб вление `mv` к к ждой строке выводит последов тельность строк, являющихся ком нд ми `mv`:

```
$ paste <(echo {1..10}.jpg | sed 's/ /\n/g') \
        <(echo {0..9}.jpg | sed 's/ /\n/g') \
  | sed 's/^/mv /'
mv 1.jpg   0.jpg
mv 2.jpg   1.jpg
⋮
mv 10.jpg  9.jpg
```

Н зн чение однострочник теперь р скрыто: он генерирует 10 ком нд для переименов ния ф йлов изобр жений с *1.jpg* по *10.jpg*. Новые имен — от *0.jpg* до *9.jpg* соответственно. Перед чу вывод в `bash` выполняют ком нды `mv`:

```
$ paste <(echo {1..10}.jpg | sed 's/ /\n/g') \
        <(echo {0..9}.jpg | sed 's/ /\n/g') \
| sed 's/^/mv /' \
| bash
```

Дерзкие однострочники похожи н головоломки, они брос ют вызов в шим творческим способностям и позволяют отт чив ть н выки.

В этой гл ве вы будете созд в ть дерзкие однострочники, подобные предыдущему примеру, ш г з ш гом, используя следующую волшебную формулу:

1. Придум йте ком нду, котор я реш ет ч сть з д чи.
2. З пустите ком нду и проверьте вывод.
3. Вызовите ком нду из истории повторно и внесите необходимые пр вки.
4. Повторяйте ш ги 2 и 3, пок в ш ком нд не д ст жел емый результ т.

Эт гл в будет н стоящей тренировкой для в шего мозг . Не удивляйтесь, что времен ми вы будете оз д чены пример ми. Двиг йтесь ш г з ш гом и з пуск йте ком нды н компьютере по мере их прочтения.

Некоторые дерзкие однострочники в этой гл ве слишком длинны для одной строки, поэтому р зделены н несколько строк обр тной косой чертой. Мы, одн ко, не ст нем н зыв ть их «дерзкими двустрочник ми» или «дерзкими семистрочник ми».

Приготовьтесь быть дерзкими

Прежде чем вы н чнете созд в ть дерзкие однострочники, н йдите время, чтобы н строиться н пр вильный л д:

- Будьте гибкими.
- Подум йте, с чего н ч ть.
- Изучите инструменты тестиров ния.

Р ссмотрим к ждую из этих идей по очереди.

Будьте гибкими

Ключом к н пис нию дерзких однострочников является *гибкость*. К этому моменту вы изучили несколько з меч тельных инструментов — б зовый н бор прогр мм Linux (и множество способов их з пуск), т кже использов ние истории ком нд, ред ктиров ние ком ндной строки и многое другое. Вы можете комбиниров ть эти инструменты р зными способ ми, и у к ждой з д чи обычно есть несколько решений.

Д же с мые простые з д чи Linux можно решить р зными способ ми. Подум йте, к к вывести н экр н все ф йлы *.jpg* в в шем текущем к т логе. Готов поспорить, что 99,9% пользов телей Linux з пустили бы т кую ком нду:

```
$ ls *.jpg
```

Но это лишь одно из многих решений. Н пример, вы можете перечислить все ф йлы в к т логе, з тем использов ть `grep` для поиск только тех имен, которые з к нчив ются н *.jpg*:

```
$ ls | grep '\.jpg$'
```

Почему стоит выбр ть именно это решение? Вы видели пример в р зделе «Длинные списки ргументов» н с. 152, когд к т лог содерж л т к много ф йлов, что их нельзя было перечислить с помощью сопост вления с ш блоном. Техник поиск р сширения имени ф йл является н дежным общим подходом к решению всех видов з д ч. Но в жно быть гибким и поним ть свои инструменты, чтобы

вы могли применить лучший из них в нужное время. Можно ск з ть, что это некое волшебство, которое вы используете при созд нии дерзких однострочников.

Все следующие ком нды отобр ж ют ф йлы *.jpg* в текущем к т логе. Попробуйте р зобр ться, к к р бот ет к жд я ком нд :

```
$ echo $(ls *.jpg)
$ bash -c 'ls *.jpg'
$ cat <(ls *.jpg)
$ find . -maxdepth 1 -type f -name \*.jpg -print
$ ls > tmp && grep '\.jpg$' tmp && rm -f tmp
$ paste <(echo ls) <(echo \*.jpg) | bash
$ bash -c 'exec $(paste <(echo ls) <(echo \*.jpg))'
$ echo 'monkey *.jpg' | sed 's/monkey/ls/' | bash
$ python -c 'import os; os.system("ls *.jpg")'
```

Один ковы ли результ ты? Можно ли придум ть другие в ри нты ком нд для этой з д чи?

Подумайте, с чего начать

К ждый дерзкий однострочник н чин ется с вывод простой ком нды. Этот вывод может быть содержимым ф йл , ч стью ф йл , списком к т логов, последов тельностью цифр или букв, списком пользов телей, д той и временем или другими д нными. Т ким обр зом, в ш перв я з д ч состоит в том, чтобы определить исходные д нные для в шей ком нды.

Н пример, если вы хотите узн ть 17-ю букву нглийского лф вит , в ши исходные д нные могут состоять из 26 букв, полученных путем р сширения фигурных скобок:

```
$ echo {A..Z}
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
```

Когд вы получите этот результ т, следующим ш гом будет решить, к к м нипулиров ть этими д нными, чтобы достичь в шей цели. В м нужно р зделить вывод н строки или столбцы? Соединить вывод с другой информ цией? Преобр зов ть вывод более сложным способом? Чтобы выполнить это з д ние, обр титесь к прогр мм м из гл в 1 и 5, н пример `grep`, `sed` и `cut`, и примените их, используя приемы из гл вы 7.

В этом примере вы можете н печ т ть 17-е поле с помощью `awk` или уд лить пробелы с помощью `sed` и н йти 17-й символ с помощью `cut`:

```
$ echo {A..Z} | awk '{print $(17)}'
Q
```

```
$ echo {A..Z} | sed 's/ //g' | cut -c17
Q
```

Если вы хотите н печ т ть месяцы год , в ши исходные д нные могут быть числ ми от 1 до 12, полученными т кже с помощью р сширения фигурных скобок:

```
$ echo {1..12}
1 2 3 4 5 6 7 8 9 10 11 12
```

Д лее измените р сширение фигурной скобки, чтобы оно формиров ло д ты для первого дня к ждого месяц (с 01.01.2021 по 01.12.2021). З тем з пустите `date -d` в к ждой строке, чтобы получить н зв ния месяцев:

```
$ echo 2021-{01..12}-01 | xargs -n1 date +%B -d
January
February
March
⋮
December
```

Или предположим, что вы хотите узн ть количество символов в с мом длинном имени ф йл в текущем к т логе. В ши исходные д нные могут быть списком к т логов:

```
$ ls
animals.txt cartoon-mascots.txt ... zebra-stripes.txt
```

Используйте `awk` для подсчет символов в к ждом имени ф йл с помощью `wc -c`:

```
$ ls | awk '{print "echo -n", $0, "| wc -c"}'
echo -n "animals.txt" | wc -c
echo -n "cartoon-mascots.txt | wc -c"
⋮
echo -n "zebra-stripes.txt | wc -c"
```

Здесь п р метр `-n` не позволяет ком нде `echo` печ т ть символы новой строки, которые меш ли бы пр вильному подсчету. Н конец, перед йте ком нды в `bash` для их з пуск , отсортируйте числовые результ ты от большего к меньшему и выведите м ксим льное (первое) зн чение с помощью `head -n1`:

```
$ ls | awk '{print "echo -n", $0, "| wc -c"}' | bash | sort -nr | head -n1
23
```

Последний пример был сложным, т к к к решение включ ет конвейеры, генерирующие строки с перед чей их следующему конвейеру. Тем не менее общий принцип тот же: определитесь с исходными д нными и м нипулируйте ими в соответствии со своими з д ч ми.

Изучите инструменты тестирования

Созд ние дерзкого однострочник может быть сопряжено со многими проб ми и ошибк ми. Следующие инструменты и методы помогут в м быстро опробов ть р зличные решения.

Используйте историю ком нд и ред ктиров ние ком ндной строки.

Не вводите ком нды повторно во время эксперимент . Используйте приемы из гл вы 3, чтобы вызыв ть предыдущие ком нды, н стр ив ть их и з пуск ть.

Доб вляйте `echo`*, чтобы проверить свои выр жения.*

Если вы не уверены, к к будет вычислено выр жение, используйте ком нду `echo`, чтобы увидеть результ ты вычислений в ст нд ртном выводе.

Используйте `ls` *или доб вьте* `echo` *для проверки деструктивных ком нд.*

Если в ш ком нд вызыв ет `rm`, `mv`, `cp` или другие ком нды, которые могут перез писыв ть или уд лять ф йлы, поместите перед ними `echo`, чтобы контролиров ть, к кие ф йлы будут з тронуты (н пример, вместо `rm` выполните ком нду `echo rm`). Еще один способ обезоп сить себя — з менить `rm` н `ls` для получения списк ф йлов, которые будут уд лены.

Вст вляйте `tee` *для просмотр промежуточных результ тов.*

Если вы хотите контролиров ть вывод (*stdout*) в середине длинного конвейер , вст вьте ком нду `tee`, чтобы сохр нить вывод в ф йл. Следующ я ком нд сохр няет вывод `command3` в ф йле `outfile`, одновременно перен пр вляя его в `command4`:

```
$ command1 | command2 | command3 | tee outfile | command4 | command5
$ less outfile
```

Теперь д в йте созд дим несколько дерзких однострочников!

Вставка имени файла в последовательность

Этот однострочник похож н тот, что был в н ч ле гл вы (переименов ние ф йлов *.jpg*), но более дет лизиров н. И это т кже ре льн я ситу ция, с которой я столкнулся, когд пис л эту книгу. К к и предыдущий однострочник, он сочет ет в себе дв метод из гл вы 7: подст новку процесс и перед чу ком нды в `bash` через ст нд ртный ввод. В результ те можно получить ш блон для решения подобных з д ч.

Я н пис л эту книгу н компьютере с Linux, используя язык н бор текст AsciiDoc (https://asciidoc.org/). Дет ли язык здесь нев жны, в жно то, что к жд я гл в был отдельным ф йлом, и изн ч льно их было 10:

```
$ ls
ch01.asciidoc ch03.asciidoc ch05.asciidoc ch07.asciidoc ch09.asciidoc
ch02.asciidoc ch04.asciidoc ch06.asciidoc ch08.asciidoc ch10.asciidoc
```

В к кой-то момент я решил вст вить дополнительную гл ву между второй и третьей. Это требов ло переименов ния некоторых ф йлов. Гл вы 3–10 ст новились 4–11, перед ними доб влял сь нов я гл в 3 (*ch03.asciidoc*). Я мог бы переименов ть ф йлы вручную, н чин я с *ch11.asciidoc* и двиг ясь в обр тном н пр влении[1]:

```
$ mv ch10.asciidoc ch11.asciidoc
$ mv ch09.asciidoc ch10.asciidoc
$ mv ch08.asciidoc ch09.asciidoc
⋮
$ mv ch03.asciidoc ch04.asciidoc
```

Но этот метод утомителен (предст вьте, если бы было 1000 ф йлов вместо 11!), и вместо этого я н пис л необходимые ком нды `mv` и перед л их в `bash`. Вним тельно посмотрите н предыдущие ком нды `mv` и подум йте, к к бы вы их н пис ли.

Н чнем с исходных имен ф йлов от *ch03.asciidoc* до *ch10.asciidoc*. Их можно н печ т ть, используя р сширение фигурных скобок, т кое к к `ch{10..03}.asciidoc`, к к в первом примере в этой гл ве, но для большей гибкости используем ком нду `seq -w` для вывод чисел:

```
$ seq -w 10 -1 3
10
09
08
⋮
03
```

З тем превр тим эту числовую последов тельность в имен ф йлов, перед в ее в `sed`:

```
$ seq -w 10 -1 3 | sed 's/\(.*\)/ch\1.asciidoc/'
ch10.asciidoc
ch09.asciidoc
⋮
ch03.asciidoc
```

[1] Н чин ть с *ch03.asciidoc* и двиг ться в прямом н пр влении было бы оп сно, поним ете, почему? Если нет, созд йте эти ф йлы с помощью ком нды `touch ch{01..10}.asciidoc` и попробуйте с ми.

Теперь у н с есть список исходных имен ф йлов. Сдел ем то же с мое для гл в 4–11, чтобы созд ть требуемые имен ф йлов:

```
$ seq -w 11 -1 4 | sed 's/\(.*\)/ch\1.asciidoc/'
ch11.asciidoc
ch10.asciidoc
⋮
ch04.asciidoc
```

Чтобы сформиров ть ком нды `mv`, нужно н печ т ть исходное и новое имен ф йлов рядом. В первом примере в этой гл ве проблем «н печ т ть рядом» реш л сь с помощью `paste` и использов л сь подст новк процесс для обр ботки двух н печ т нных списков к к ф йлов. Здесь сдел ем т к же:

```
$ paste <(seq -w 10 -1 3 | sed 's/\(.*\)/ch\1.asciidoc/') \
        <(seq -w 11 -1 4 | sed 's/\(.*\)/ch\1.asciidoc/')
ch10.asciidoc ch11.asciidoc
ch09.asciidoc ch10.asciidoc
⋮
ch03.asciidoc ch04.asciidoc
```

Предыдущ я ком нд выглядит т к, словно в м нужно н печ т ть очень много текст , но, используя историю ком нд и ред ктиров ние ком нд- ной строки в стиле Emacs, этого можно избеж ть. Чтобы перейти от простой ком нды `seq` и `sed` к ком нде `paste`, необходимо выполнить следующее:

1. Вызв ть предыдущую ком нду из истории с помощью стрелки вверх.
2. Н ж ть Ctrl-A, з тем Ctrl-K, чтобы вырез ть всю строку.
3. Ввести слово `paste`, з тем пробел.
4. Дв жды н ж ть Ctrl-Y, чтобы созд ть две копии ком нд `seq` и `sed`.
5. Использов ть сочет ния кл виш для перемещения и ред ктиров ния, чтобы изменить вторую копию.

И т к д лее.

Доб вляйте `mv` к к ждой строке, н пр вляя вывод в `sed`:

```
$ paste <(seq -w 10 -1 3 | sed 's/\(.*\)/ch\1.asciidoc/') \
        <(seq -w 11 -1 4 | sed 's/\(.*\)/ch\1.asciidoc/') \
  | sed 's/^/mv /'
mv ch10.asciidoc     ch11.asciidoc
mv ch09.asciidoc     ch10.asciidoc
⋮
mv ch03.asciidoc     ch04.asciidoc
```

В к честве последнего ш г перед йте ком нды в bash для выполнения:

```
$ paste <(seq -w 10 -1 3 | sed 's/\(.*\)/ch\1.asciidoc/') \
        <(seq -w 11 -1 4 | sed 's/\(.*\)/ch\1.asciidoc/') \
  | sed 's/^/mv /' \
  | bash
```

Я использов л именно это решение для своей книги. После выполнения ком нд mv итоговыми ф йл ми ст ли гл вы 1, 2 и 4–11, ост вив место для новой гл вы 3:

```
$ ls ch*.asciidoc
ch01.asciidoc  ch04.asciidoc  ch06.asciidoc  ch08.asciidoc  ch10.asciidoc
ch02.asciidoc  ch05.asciidoc  ch07.asciidoc  ch09.asciidoc  ch11.asciidoc
```

Опис нный ш блон действий можно использов ть во всех ситу циях для з пуск последов тельности связ нных ком нд:

1. Перед йте ргументы ком нды в виде списков в ст нд ртный вывод.
2. Выведите списки столбц ми с помощью paste и подст новки процесс .
3. Доб вьте sed перед именем ком нды, з менив символ н ч л строки (^) именем прогр ммы и пробелом.
4. Перед йте результ ты в bash.

Проверка совпадающих пар файлов

Этот однострочник вдохновлен пр ктикой использов ния Mediawiki, прогр ммного обеспечения, н котором р бот ет Wikipedia и тысячи других вики. Mediawiki позволяет пользов телям з груж ть изобр жения для просмотр . Большинство пользов телей дел ют это вручную через веб-формы: н жим ют «Выбр ть ф йл», чтобы открыть ди логовое окно, выбир ют ф йл, доб вляют небольшое опис ние в форму и н жим ют «З грузить». Администр торы вики используют сцен рий, который считыв ет весь к т лог и з груж ет изобр жения из него. К ждый ф йл изобр жения (ск жем, *bald_eagle.jpg*) связ н с текстовым ф йлом (*bald_eagle.txt*), содерж щим опис ние изобр жения.

Предст вьте, что имеется к т лог с сотнями ф йлов изобр жений и текст . Вы хотите убедиться, что к ждому ф йлу изобр жения соответствует текстовый ф йл, и н оборот. Вот уменьшенн я версия этого к т лог :

```
$ ls
bald_eagle.jpg blue_jay.jpg cardinal.txt robin.jpg wren.jpg
bald_eagle.txt cardinal.jpg oriole.txt robin.txt wren.txt
```

Д в йте придум ем дв р зных решения для выявления несоответствий ф йлов. Для первого решения созд дим дв список : для ф йлов JPEG и текстовых ф йлов. Потом используем `cut`, чтобы уд лить р сширения *.txt* и *.jpg*:

```
$ ls *.jpg | cut -d. -f1
bald_eagle
blue_jay
cardinal
robin
wren
$ ls *.txt | cut -d. -f1
bald_eagle
cardinal
oriole
robin
wren
```

З тем ср вним списки с помощью `diff`, используя подст новку процесс :

```
$ diff <( ls *.jpg | cut -d. -f1 ) <( ls *.txt | cut -d. -f1 )
2d1
< blue_jay
3a3
> oriole
```

Н этом можно было бы ост новиться, потому что в выводе ук з но, что в первом списке есть неп рный ф йл *blue_jay* (т. е. *blue_jay.jpg*), во втором списке — *oriole* (т. е. *oriole.txt*). Тем не менее д в йте сдел ем результ ты более информ тивными. Уд лим ненужные строки, выполнив поиск символов < и > в н ч ле к ждой строки:

```
$ diff <(ls *.jpg | cut -d. -f1) <(ls *.txt | cut -d. -f1) \
  | grep '^[<>]'
< blue_jay
> oriole
```

З тем используем `awk`, чтобы доб вить пр вильное р сширение к к ждому имени ф йл (`$2`), в з висимости от того, к кой символ предшествует имени ф йл , < или >:

```
$ diff <(ls *.jpg | cut -d. -f1) <(ls *.txt | cut -d. -f1) \
  | grep '^[<>]' \
  | awk '/^</{print $2 ".jpg"} /^>/{print $2 ".txt"}'
blue_jay.jpg
oriole.txt
```

Теперь у н с есть список неп рных ф йлов. Одн ко в этом решении есть небольш я ошибк . Предположим, что текущий к т лог содержит имя ф йл

yellow.canary.jpg, в котором есть две точки. Предыдущ я ком нд выд ст непр вильный результ т:

```
blue_jay.jpg
oriole.txt
yellow.jpg                    Это неверно
```

Эт проблем возник ет из-з того, что две ком нды `cut` уд ляют символы, н чин я с первой, не последней точки, поэтому *yellow.canary.jpg* усек ется до *yellow*, не до *yellow.canary*. Для решения проблемы з меним `cut` н `sed`, чтобы уд лить символы от последней точки до конц строки:

```
$ diff <(ls *.jpg | sed 's/\.[^.]*$//') \
       <(ls *.txt | sed 's/\.[^.]*$//') \
  | grep '^[<>]' \
  | awk '/</{print $2 ".jpg"} />/{print $2 ".txt"}'
blue_jay.txt
oriole.jpg
yellow.canary.txt
```

Первое решение готово.

Второе решение использует другой подход. Вместо применения `diff` к двум списк м созд дим один список и отсеем совп д ющие п ры имен ф йлов. Н чнем с уд ления р сширений ф йлов с помощью `sed` (используя тот же сцен рий `sed`, что и р ньше) и подсчит ем количество вхождений к ждой строки с помощью `uniq -c`:

```
$ ls *.{jpg,txt} \
  | sed 's/\.[^.]*$//' \
  | uniq -c
      2 bald_eagle
      1 blue_jay
      2 cardinal
      1 oriole
      2 robin
      2 wren
      1 yellow.canary
```

К жд я строк вывод содержит либо число `2`, предст вляющее совп д ющую п ру имен ф йлов, либо `1`, предст вляющее несовп д ющее имя ф йл . Используем `awk`, чтобы изолиров ть строки, н чин ющиеся с пробел и 1, и вывести только второе поле:

```
$ ls *.{jpg,txt} \
  | sed 's/\.[^.]*$//' \
  | uniq -c \
  | awk '/^ *1 /{print $2}'
```

```
blue_jay
oriole
yellow.canary
```

К к теперь доб вить отсутствующие р сширения ф йлов? Не нужно сложных м нипуляций со строк ми, просто используем `ls`, чтобы получить список ф йлов в текущем к т логе. Вст вим звездочку (*) в конец к ждой строки вывод с помощью `awk`:

```
$ ls *.{jpg,txt} \
  | sed 's/\.[^.]*$//' \
  | uniq -c \
  | awk '/^ *1 /{print $2 "*" }'
blue_jay*
oriole*
yellow.canary*
```

З тем перед дим строки в `ls` через подст новку ком нд. Оболочк выполнит сопост вление с ш блоном, и `ls` перечислит несопост вленные имен ф йлов. Готово!

```
$ ls -1 $( ls *.{jpg,txt} \
  | sed 's/\.[^.]*$//' \
  | uniq -c \
  | awk '/^ *1 /{print $2 "*"}' )
blue_jay.jpg
oriole.txt
yellow.canary.jpg
```

Создание CDPATH из вашего домашнего каталога

В р зделе «Орг низуйте свой дом шний к т лог для быстрой н виг ции» н с. 77 мы вручную н пис ли сложное выр жение для переменной `CDPATH`. Оно н чин ется с `$HOME`, з которым следуют все подк т логи `$HOME`, и з к нчив ется относительным путем `..` (родительский к т лог):

```
CDPATH=$HOME:$HOME/Work:$HOME/Family:$HOME/Finances:$HOME/Linux:$HOME/Music:..
```

Д в йте р зр бот ем однострочник для втом тического созд ния этой строки `CDPATH`, чтобы ее можно было вст вить в ф йл конфигур ции `bash`. Н чнем со списк подк т логов в `$HOME`, используя подоболочку, чтобы ком нд `cd` не могл изменить текущий к т лог оболочки:

```
$ (cd && ls -d */)
Family/ Finances/ Linux/ Music/ Work/
```

Доб вим $HOME/ перед к ждым к т логом с помощью sed:

```
$ (cd && ls -d */) | sed 's/^/$HOME\//g'
$HOME/Family/
$HOME/Finances/
$HOME/Linux/
$HOME/Music/
$HOME/Work/
```

Синт ксис предыдущего сцен рия sed усложнен, поскольку строк з мены $HOME/ содержит слеш, подст новки sed т кже используют косую черту в к честве р зделителя. Вот почему один слеш экр ниров н: $HOME\/. Вспомните из р здел «Подст новк и символ слеш » н с. 115, что sed приним ет любой символ в к честве р зделителя. Д в йте использов ть @ вместо косой черты, тогд экр ниров ние не потребуется:

```
$ (cd && ls -d */) | sed 's@^@$HOME/@g'
$HOME/Family/
$HOME/Finances/
$HOME/Linux/
$HOME/Music/
$HOME/Work/
```

З тем уберем последнюю косую черту еще одним выр жением sed:

```
$ (cd && ls -d */) | sed -e 's@^@$HOME/@' -e 's@/$@@'
$HOME/Family
$HOME/Finances
$HOME/Linux
$HOME/Music
$HOME/Work
```

Выведем результ т в одной строке, используя echo и подст новку ком нд. Обр тите вним ние, что н м больше не нужны круглые скобки вокруг cd и ls для явного созд ния подоболочки, потому что подст новк ком нд созд ет собственную подоболочку:

```
$ echo $(cd && ls -d */ | sed -e 's@^@$HOME/@' -e 's@/$@@')
$HOME/Family $HOME/Finances $HOME/Linux $HOME/Music $HOME/Work
```

Доб вим в н ч ло к т лог $HOME, в конец родительский к т лог ..:

```
echo '$HOME' \
     $(cd && ls -d */ | sed -e 's@^@$HOME/@' -e 's@/$@@') \
     ..
$HOME $HOME/Family $HOME/Finances $HOME/Linux $HOME/Music $HOME/Work ..
```

З меним пробелы н двоеточия, перен пр вив весь вывод в ком нду tr:

```
$ echo '$HOME' \
       $(cd && ls -d */ | sed -e 's@^@$HOME/@' -e 's@/$@@') \
       .. \
  | tr ' ' ':'
$HOME:$HOME/Family:$HOME/Finances:$HOME/Linux:$HOME/Music:$HOME/Work:..
```

Н конец, доб вим переменную окружения `CDPATH` — и определение переменной для вст вки в ф йл конфигур ции `bash` созд но. Сохр ним эту ком нду в сце-н рии, чтобы генериров ть строку в любое время, н пример когд доб вляем в `$HOME` новый подк т лог:

```
$ echo 'CDPATH=$HOME' \
$(cd && ls -d */ | sed -e 's@^@$HOME/@' -e 's@/$@@') \
.. \
| tr ' ' ':'
CDPATH=$HOME:$HOME/Family:$HOME/Finances:$HOME/Linux:$HOME/Music:$HOME/Work:..
```

Создание тестовых файлов

Р спростр ненной з д чей при р зр ботке прогр ммного обеспечения является тестиров ние — перед ч в прогр мму с мых р знообр зных д нных для про-верки того, что прогр мм ведет себя к к з дум но. Следующий однострочник генерирует тысячу ф йлов, содерж щих случ йный текст; эти ф йлы можно использов ть при тестиров нии прогр ммного обеспечения. Тысяч — условное число, вы можете генериров ть столько ф йлов, сколько з хотите.

Однострочник будет случ йным обр зом выбир ть слов из большого текстового ф йл и созд в ть тысячу ф йлов меньшего р змер со случ йным содержимым. Иде льным исходным ф йлом является системный слов рь */usr/share/dict/words*, содерж щий 102 305 слов, к ждое из которых н ходится в отдельной строке.

```
$ wc -l /usr/share/dict/words
102305 /usr/share/dict/words
```

Чтобы созд ть этот дерзкий однострочник, н м нужно решить четыре головоломки:

1. Случ йное перемешив ние ф йл слов ря.
2. Выбор случ йного количеств строк из ф йл слов ря.
3. Созд ние выходного ф йл для хр нения результ тов.
4. З пуск полученной ком нды тысячу р з.

Чтобы перемеш ть слов в случ йном порядке, используем ком нду `shuf`. При к ждом выполнении ком нды `shuf /usr/share/dict/words` выводится более

ст тысяч строк, поэтому просмотрим первые несколько случ йных строк с помощью ком нды head:

```
$ shuf /usr/share/dict/words | head -n3
evermore
shirttail
tertiary
$ shuf /usr/share/dict/words | head -n3
interactively
opt
perjurer
```

Перв я головоломк решен . Теперь о том, к к выбр ть случ йное количество строк из перемеш нного слов ря. shuf имеет п р метр -n для печ ти з д нного количеств строк, но мы хотим, чтобы зн чение изменялось для к ждого созд в емого выходного ф йл . К сч стью, в bash есть переменн я RANDOM, котор я содержит случ йное положительное целое число от 0 до 32 767. Его зн чение меняется к ждый р з, когд мы обр щ емся к переменной:

```
$ echo $RANDOM $RANDOM $RANDOM
7855 11134 262
```

Поэтому з пустим shuf с п р метром -n $RANDOM, чтобы вывести случ йное количество случ йных строк. Опять же, полный вывод может быть очень длинным, поэтому перед дим результ ты wc -l, чтобы убедиться, что количество строк меняется при к ждом выполнении:

```
$ shuf -n $RANDOM /usr/share/dict/words | wc -l
9922
$ shuf -n $RANDOM /usr/share/dict/words | wc -l
32465
```

Мы решили вторую головоломку. Д лее потребуется тысяч выходных ф йлов, точнее, тысяч ф йлов с р зными имен ми. Чтобы сгенериров ть имен ф йлов, з пустим прогр мму pwgen, котор я генерирует случ йные строки букв и цифр:

```
$ pwgen
eng9nooG ier6YeVu AhZ7naeG Ap3quail poo2Ooj9 OYiuri9m iQuash0E voo3Eph1
IeQu7mi6 eipaC2ti exah8iNg oeGhahm8 airooJ8N eiZ7neez Dah8Vooj dixiV1fu
Xiejoti6 ieshei2K iX4isohk Ohm5gaol Ri9ah4eX Aiv1ahg3 Shaew3ko zohB4geu
⋮
```

Доб вим п р метр -N1, чтобы сгенериров ть только одну строку, и ук жем длину строки (10) в к честве ргумент :

```
$ pwgen -N1 10
ieb2ESheiw
```

При необходимости сдел ем строку более похожей н имя текстового ф йл , используя подст новку ком нд:

```
$ echo $(pwgen -N1 10).txt
ohTie8aifo.txt
```

Третья головоломк решен ! Теперь у н с есть все необходимое для созд ния одного случ йного текстового ф йл . Используем опцию `-o` ком нды `shuf`, чтобы сохр нить результ т в ф йле:

```
$ mkdir -p /tmp/randomfiles && cd /tmp/randomfiles
$ shuf -n $RANDOM -o $(pwgen -N1 10).txt /usr/share/dict/words
```

и проверим результ ты:

```
$ ls                          Вывести новый файл
Ahxiedie2f.txt
$ wc -l Ahxiedie2f.txt        Сколько строк он содержит?
13544 Ahxiedie2f.txt
$ head -n3 Ahxiedie2f.txt     Посмотреть первые несколько строк
saviors
guerillas
forecaster
```

Выглядит неплохо! Последняя головоломк состоит в выполнении предыдущей ком нды `shuf` тысячу р з. Конечно, можно использов ть цикл:

```
for i in {1..1000}; do
  shuf -n $RANDOM -o $(pwgen -N1 10).txt /usr/share/dict/words
done
```

но это не т к весело, к к созд ние дерзкого однострочник . Вместо этого д в йте з р нее сгенерируем ком нды в виде строк и перед дим их в `bash`. В к честве тест н печ т ем жел емую ком нду один р з с помощью `echo`. Доб вим оди-н рные к вычки, чтобы г р нтиров ть, что `$RANDOM` не вычисляется и `pwgen` не з пуск ется:

```
$ echo 'shuf -n $RANDOM -o $(pwgen -N1 10).txt /usr/share/dict/words'
shuf -n $RANDOM -o $(pwgen -N1 10).txt /usr/share/dict/words
```

Эту ком нду можно перед ть в `bash` для выполнения:

```
$ echo 'shuf -n $RANDOM -o $(pwgen -N1 10).txt /usr/share/dict/words' | bash
$ ls
eiFohpies1.txt
```

Теперь повторим это тысячу р з, используя ком нду `yes`, перед нную в `head`, з тем отпр вим результ ты в `bash` — и мы решили четвертую головоломку:

```
$ yes 'shuf -n $RANDOM -o $(pwgen -N1 10).txt /usr/share/dict/words' \
  | head -n 1000 \
  | bash
$ ls
Aen1lee0ir.txt IeKaveixa6.txt ahDee9lah2.txt paeR1Poh3d.txt
Ahxiedie2f.txt Kas8ooJahK.txt aoc0Yoohoh.txt sohl7Nohho.txt
CudieNgee4.txt Oe5ophae8e.txt haiV9mahNg.txt uchiek3Eew.txt
⋮
```

Если вы предпочитаете тысячу случайных файлов изображений вместо текстовых файлов, используйте ту же технику (`yes`, `head` и `bash`) и замените `shuf` командой, которая генерирует случайное изображение. Вот дорабатываемый дерзкий однострочник из решения Марка Сетчелла (Mark Setchell) в Stack Overflow (https://stackoverflow.com/questions/29011391/generate-random-bmp-in-cli). Он запускает команду `convert` из графического пакета *ImageMagick* для создания случайных изображений размером 100×100 пикселей, состоящих из разноцветных квадратов:

```
$ yes 'convert -size 8x8 xc: +noise Random -scale 100x100 $(pwgen -N1 10).png' \
  | head -n 1000 \
  | bash
$ ls
Bahdo4Yaop.png Um8ju8gie5.png aing1QuaiX.png ohi4ziNuwo.png
Eem5leijae.png Va7ohchiep.png eiMoog1kou.png ohnohwu4Ei.png
Eozaing1ie.png Zaev4Quien.png hiecima2Ye.png quaepaiY9t.png
⋮
$ display Bahdo4Yaop.png          Просмотр первой картинки
```

Создание пустых файлов

Иногда все, что вам нужно для тестирования, — это множество файлов с разными именами, даже если они пусты. Создать тысячу пустых файлов с именами от *file0001.txt* до *file1000.txt* очень просто:

```
$ mkdir /tmp/empties Create a directory for the files
$ cd /tmp/empties
$ touch file{01..1000}.txt Generate the files
```

Если вы предпочитаете более интересные имена файлов, выберите их случайным образом из системного словаря. Используйте `grep`, чтобы ограничить имена строчными буквами для простоты (избегая пробелов, апострофов и других символов, которые были бы специфичны для оболочки):

```
$ grep '^[a-z]*$' /usr/share/dict/words
a
aardvark
aardvarks
⋮
```

Перемеш йте имен с помощью shuf и выведите первую тысячу с помощью head:

```
$ grep '^[a-z]*$' /usr/share/dict/words | shuf | head -n1000
triplicating
quadruplicates
podiatrists
⋮
```

Н конец, перед йте результ ты в xargs, чтобы созд ть ф йлы с помощью touch:

```
$ grep '^[a-z]*$' /usr/share/dict/words | shuf | head -n1000 | xargs touch
$ ls
abases            distinctly       magnolia         sadden
abets             distrusts        maintaining      sales
aboard            divided          malformation     salmon
⋮
```

Резюме

Н деюсь, что примеры из этой гл вы помогли в м р звить н выки н пис ния дерзких однострочников. Некоторые из них можно использов ть к к ш блоны, которые могут ок з ться полезными в р зных ситу циях.

Но не з быв йте: дерзкие однострочники — не единственное решение. Это всего лишь один из подходов к эффективной р боте в ком ндной строке. Иногд вы можете добиться большего эффект , н пис в сцен рий. В других случ ях вы н йдете лучшие решения с помощью язык прогр ммиров ния, т кого к к Perl или Python. Тем не менее н пис ние дерзких однострочников является в жным н выком для быстрого и стильного выполнения критически в жных з д ч.

ГЛАВА 9

Использование текстовых файлов

Обычный текст является н иболее р спростр ненным форм том д нных в систем х Linux. Содержимое, перед в емое между ком нд ми в большинстве конвейеров, предст вляет собой текст. Ф йлы исходного код прогр мм, конфигур ции системы в к т логе */etc*, т кже ф йлы HTML и Markdown — все это текстовые д нные. Письм электронной почты являются текстовыми, д же вложения сохр няются для тр нспортировки к к текст. И ф йлы для повседневных з бот, т кие к к списки покупок и личные з метки, можно хр нить в виде текст .

Одн ко сегодняшний интернет предст вляет собой меш нину из потокового удио и видео, сообщений в соци льных сетях, документов в Google Docs и Office 365, PDF-ф йлов и других мультимедийных д нных (не говоря уже о д нных, обр б тыв емых мобильными приложениями, которые скрыли понятие «ф йл» от целого поколения пользов телей). В этих условиях простые текстовые ф йлы к жутся ст ромодными.

Тем не менее любой текстовый ф йл может ст ть бог тым источником д нных, которые вы можете извлечь с помощью ком нд Linux, особенно если текст структуриров н. К жд я строк в ф йле */etc/passwd*, н пример, предст вляет пользов теля Linux и имеет семь полей, включ я имя пользов теля, его числовой идентифик тор, дом шний к т лог и т. д. Поля р зделены двоеточиями, что упрощ ет н лиз ф йл с помощью `cut -d:` или `awk -F:`. Вот, н пример, ком нд , котор я печ т ет все имен пользов телей (первое поле) в лф витном порядке:

```
$ cut -d: -f1 /etc/passwd | sort
avahi
backup
daemon
⋮
```

А теперь перейдем к ком нде, котор я отделяет пользов телей-людей от системных учетных з писей по их числовым идентифик тор м и отпр вляет пользов телям приветственное электронное письмо. Д в йте построим этот дерзкий однострочник ш г з ш гом. Во-первых, используем `awk` для вывод

имен пользов телей (поле 1), когд числовой идентифик тор пользов теля (поле 3) р вен 1000 или больше:

```
$ awk -F: '$3>=1000 {print $1}' /etc/passwd
jones
smith
```

З тем созд дим приветствие, перед в его в `xargs`:

```
$ awk -F: '$3>=1000 {print $1}' /etc/passwd \
  | xargs -I@ echo "Hi there, @!"
Hi there, jones!
Hi there, smith!
```

З тем сгенерируем ком нды (строки) для перед чи к ждого приветствия ком нде `mail`, котор я отпр вляет электронное письмо д нному пользов телю с з д нной строкой темы (`-s`):

```
$ awk -F: '$3>=1000 {print $1}' /etc/passwd \
  | xargs -I@ echo 'echo "Hi there, @!" | mail -s greetings @'
echo "Hi there, jones!" | mail -s greetings jones
echo "Hi there, smith!" | mail -s greetings smith
```

Н конец, перед ем сгенериров нные ком нды в `bash` для отпр вки электронных писем:

```
$ awk -F: '$3>=1000 {print $1}' /etc/passwd \
  | xargs -I@ echo 'echo "Hi there, @!" | mail -s greetings @' \
  | bash
echo "Hi there, jones!" | mail -s greetings jones
echo "Hi there, smith!" | mail -s greetings smith
```

Большинство решений в этой книге н чин лось с условия, что существует текстовый ф йл и его содержимое необходимо обр бот ть с помощью ком нд. Пришло время изменить этот подход и н учиться *формиров ть новые текстовые ф йлы*, которые хорошо сочет ются с ком нд ми Linux[1]. Это выигрышн я стр тегия для эффективной р боты в Linux. Всего четыре ш г позволяют добиться этого:

1. Про н лизируйте конкретную з д чу обр ботки д нных, которую хотите решить.
2. Сохр ните д нные в текстовом ф йле в удобном форм те.
3. Придум йте ком нды Linux, которые обр б тыв ют ф йл и реш ют з д чу.
4. (*Необяз тельно*) Оформите эти ком нды к к сцен рии, псевдонимы или функции, чтобы упростить их выполнение.

[1] Этот подход н логичен р зр ботке схемы б зы д нных, котор я хорошо р бот ет с известными з прос ми.

В этой гл ве вы позн комитесь со множеством структуриров нных текстовых ф йлов и созд дите ком нды для их обр ботки, чтобы решить несколько конкретных з д ч.

Первый пример: поиск файлов

Предположим, что в ш дом шний к т лог содержит десятки тысяч ф йлов и подк т логов, и время от времени вы не можете вспомнить, что где лежит. Ком нд `find` н ходит ф йл по имени, н пример *animals.txt*:

```
$ find $HOME -name animals.txt -print
/home/smith/Work/Writing/Books/Lists/animals.txt
```

Но ком нд `find` медленн я, потому что он обыскив ет весь в ш дом шний к т лог. Это ш г 1, н меч ющий пр ктическую з д чу — быстрый поиск ф йлов в в шем дом шнем к т логе по имени.

Ш г 2 — сохр нение д нных в текстовом ф йле в удобном форм те. З пустите `find`, чтобы созд ть список всех в ших ф йлов и к т логов, по одному пути к ф йлу в строке, и сохр ните его в скрытом ф йле:

```
$ find $HOME -print > $HOME/.ALLFILES
$ head -n3 $HOME/.ALLFILES
/home/smith
/home/smith/Work
/home/smith/Work/resume.pdf
⋮
```

Теперь у в с есть д нные — построчный индекс в ших ф йлов. Н ш ге 3 нужно созд ть ком нды для ускорения поиск ф йлов. Используем `grep`, т к к к быстрее просмотреть один большой ф йл, чем выполнять поиск в большом дереве к т логов:

```
$ grep animals.txt $HOME/.ALLFILES
/home/smith/Work/Writing/Books/Lists/animals.txt
```

Ш г 4 — упростить выполнение ком нды. Н пишем однострочный сцен рий с именем `ff` (*find files*) для поиск ф йлов, который з пуск ет `grep` с любыми п р метр ми, з д нными пользов телем, и строкой поиск :

Листинг 9.1. Сценарий `ff`

```
#!/bin/bash
# $@ означает "все аргументы, переданные скрипту"
grep "$@" $HOME/.ALLFILES
```

Сделаем сценарий исполняемым и поместим его в любой каталог в пути поиска, например в вашем личный подкаталог *bin*:

```
$ chmod +x ff
$ echo $PATH                          Проверьте ваш путь для поиска команд
/home/smith/bin :/usr/local/bin:/usr/bin:/bin
$ mv ff ~/bin
```

Используйте `ff`, чтобы быстро найти файлы, если не можете вспомнить, куда их положили.

```
$ ff animal
/home/smith/Work/Writing/Books/Lists/animals.txt
$ ff -i animal | less                 Нечувствительная к регистру команда grep
/home/smith/Work/Writing/Books/Lists/animals.txt
/home/smith/Vacations/Zoos/Animals/pandas.txt
/home/smith/Vacations/Zoos/Animals/tigers.txt
⋮
$ ff -i animal | wc -l                Сколько совпадений?
16
```

Время от времени перезапускайте команду `find`, чтобы обновить индекс (еще лучше — создайте плановое задание с помощью `cron`, см. «Изучите команды cron, crontab и at» на с. 225). Вуаля! Из двух небольших команд мы создали быструю и гибкую утилиту для поиска файлов. Конечно, системы Linux предоставляют другие приложения для быстрой индексации и поиска файлов, такие как команда `locate` и утилиты поиска в GNOME, KDE Plasma и других окружениях рабочего стола, но это к делу не относится. Посмотрите, как легко было разработать такой инструмент самостоятельно. И ключом к успеху было создание текстового файла.

Проверка срока действия домена

В следующем примере предположим, что вы владеете несколькими доменными именами в интернете и хотите отслеживать, когда истекает срок действия регистрации. Шаг 1 выполнен — практическая задача определена. Шаг 2 — создать файл со списком этих доменных имен, например *domains.txt*, по одному доменному имени в строке:

```
example.com
oreilly.com
efficientlinux.com
⋮
```

Шаг 3 — написать команды, которые дополнят этот текстовый файл датами истечения срока действия. Начнем с команды `whois`, которая запрашивает у регистратора информацию о домене:

```
$ whois example.com | less
Domain Name: EXAMPLE.COM
Registry Domain ID: 2336799_DOMAIN_COM-VRSN
Registrar WHOIS Server: whois.iana.org
Updated Date: 2021-08-14T07:01:44Z
Creation Date: 1995-08-14T04:00:00Z
Registry Expiry Date: 2022-08-13T04:00:00Z
⋮
```

Дате истечения срока действия предшествует строка *Registry Expiry Date*, которую можно изолировать с помощью `grep` и `awk`:

```
$ whois example.com | grep 'Registry Expiry Date:'
Registry Expiry Date: 2022-08-13T04:00:00Z
$ whois example.com | grep 'Registry Expiry Date:' | awk '{print $4}'
2022-08-13T04:00:00Z
```

Сделаем дату более читаемой с помощью команды `date --date`, которая преобразовывает строку даты из одного формата в другой:

```
$ date --date 2022-08-13T04:00:00Z
Sat Aug 13 00:00:00 EDT 2022
$ date --date 2022-08-13T04:00:00Z +'%Y-%m-%d'       формат ГГГГ-ММ-ДД
2022-08-13
```

Используем подстановку команд, чтобы передать строку даты из `whois` в `date`:

```
$ echo $(whois example.com | grep 'Registry Expiry Date:' | awk '{print $4}')
2022-08-13T04:00:00Z
$ date \
  --date $(whois example.com \
           | grep 'Registry Expiry Date:' \
           | awk '{print $4}') \
  +'%Y-%m-%d'
2022-08-13
```

Теперь у нас есть команда, которая запрашивает регистратора и выводит дату истечения срока действия. Создадим скрипт `check-expiry`, показанный в листинге 9.2, который запускает предыдущую команду и выводит на экран дату истечения срока действия и через символ табуляции — имя домена:

```
$ ./check-expiry example.com
2022-08-13      example.com
```

Листинг 9.2. check-expiry скрипт

```
#!/bin/bash
expdate=$(date \
          --date $(whois "$1" \
                   | grep 'Registry Expiry Date:' \
                   | awk '{print $4}') \
          +'%Y-%m-%d')
echo "$expdate $1"        # Два значения, разделенные символом табуляции
```

Теперь проверим все домены в ф йле *domains.txt*, используя цикл. Созд дим новый сцен рий `check-expiry-all`, к к пок з но в листинге 9.3.

Листинг 9.3. check-expiry-all сценарий

```
#!/bin/bash
cat domains.txt | while read domain; do
    ./check-expiry "$domain"
    sleep 5               # Будьте снисходительны к серверу регистратора
done
```

З пустим сцен рий в фоновом режиме, т к к к выполнение может з нять некоторое время, если у в с много доменов, и перен пр вим весь вывод (*stdout* и *stderr*) в ф йл:

```
$ ./check-expiry-all &> expiry.txt &
```

Когд сцен рий з вершится, ф йл *expiry.txt* будет содерж ть нужную информ цию:

```
$ cat expiry.txt
2022-08-13    example.com
2022-05-26    oreilly.com
2022-09-17    efficientlinux.com
⋮
```

Ур ! Но не ост н влив йтесь н достигнутом. Ф йл *expiry.txt* хорошо структуриров н для д льнейшей обр ботки, т к к содержит дв столбц , р зделенные символом т буляции. Можно, н пример, отсортиров ть д ты и н йти следующий домен, требующий продления:

```
$ sort -n expiry.txt | head -n1
2022-05-26      oreilly.com
```

Или можно использов ть `awk`, чтобы н йти домены, срок действия которых (поле 1) истек или истек ет сегодня, то есть меньше или р вен сегодняшней д те (`date +%Y-%m-%d`):

```
$ awk "\$1<=\"$(date +%Y-%m-%d)\"" expiry.txt
```

Несколько з меч ний к предыдущей ком нде awk:

1. Экр ниров ние зн к долл р (перед $1) и двойные к вычки вокруг строки д ты нужны, чтобы оболочк не вычислил их до того, к к это сможет сдел ть awk.
2. Мы немного схитрили, используя строковый опер тор <= для ср внения д т. Это не м тем тическое ср внение, просто ср внение строк, но оно р бот ет, потому что форм т д ты *ГГГГ-ММ-ДД* сортируется один ково к к в лф витном, т к и в хронологическом порядк х.

Приложив немного усилий, можно было бы вычислять д ты в awk, чтобы предупрежд ть об истечении срок действия, ск жем, з две недели, з тем созд ть з пл ниров нное з д ние для з пуск скрипт к ждую ночь и отпр вки в м отчет по электронной почте. Не стесняйтесь экспериментиров ть. Суть здесь, одн ко, в том, что с помощью нескольких ком нд мы созд ли полезную утилиту, р бот ющую с текстовым ф йлом.

Создание базы данных телефонных кодов

В следующем примере используется ф йл с тремя полями, которые можно обр - б тыв ть р зными способ ми. Ф йл с именем *areacodes.txt* содержит телефонные коды городов США. Используйте один из дополнительных м тери лов к этой книге в к т логе *Chapter09/build_area_code_database* (https://resources.oreilly.com/examples/0636920601098) или созд йте свой собственный ф йл, н пример с помощью «Википедии»[1]:

```
201     NJ   Hackensack, Jersey City
202     DC   Washington
203     CT   New Haven, Stamford
⋮
989     MI   Saginaw
```

Первыми р сположите поля предск зуемой длины, чтобы столбцы выглядели ккур тно выровненными. Посмотрите, к ким сумбурным получится ф йл, если вы поместите н зв ния городов в первую колонку:

```
Hackensack, Jersey City 201 NJ
Washington     202      DC
⋮
```

[1] В офици льном списке кодов городов в форм те CSV, сост вленном Администр цией Северомерик нского пл н нумер ции, отсутствуют н зв ния городов (https://oreil.ly/SptWL).

Когда файл будет готов, можно посмотреть коды городов по штатам с помощью `grep`, добавив параметр `-w` для соответствия только полным словам (на случай, если другой текст случайно содержит *NJ*):

```
$ grep -w NJ areacodes.txt
201     NJ      Hackensack, Jersey City
551     NJ      Hackensack, Jersey City
609     NJ      Atlantic City, Trenton, southeast and central west
⋮
```

или найти город по коду:

```
$ grep -w 202 areacodes.txt
202     DC      Washington
```

либо по любой строке в файле:

```
$ grep Washing areacodes.txt
202     DC      Washington
227     MD      Silver Spring, Washington suburbs, Frederick
240     MD      Silver Spring, Washington suburbs, Frederick
⋮
```

Подсчитаем количество кодов городов с помощью `wc`:

```
$ wc -l areacodes.txt
375 areacodes.txt
```

Найдем штаты с наибольшим количеством кодов городов (победителем стала Калифорния с 38 кодами):

```
$ cut -f2 areacodes.txt | sort | uniq -c | sort -nr | head -n1
     38 CA
```

Преобразуем файл в формат CSV для импорта в табличный процессор. Выведем третье поле, заключенное в двойные кавычки, чтобы его запятые не интерпретировались как символы-разделители CSV:

```
$ awk -F'\t' '{printf "%s,%s,\"%s\"\n", $1, $2, $3}' areacodes.txt \
  > areacodes.csv
$ head -n3 areacodes.csv
201,NJ,"Hackensack, Jersey City"
202,DC,"Washington"
203,CT,"New Haven, Stamford"
```

Объединим все коды городов для данного штата в одну строку:

```
$ awk '$2~/^NJ$/{ac=ac FS $1} END {print "NJ:" ac}' areacodes.txt
NJ: 201 551 609 732 848 856 862 908 973
```

или сдел ем все то же с мое для к ждого шт т , используя м ссивы и циклы `for`, к к в р зделе «Улучшенный способ обн ружения дублик тов ф йлов» н с. 112:

```
$ awk '{arr[$2]=arr[$2] " " $1} \
         END {for (i in arr) print i ":" arr[i]}' areacodes.txt \
  | sort
AB: 403 780
AK: 907
AL: 205 251 256 334 659
⋮
WY: 307
```

Превр тим любую из предыдущих ком нд в псевдонимы, функции или сцен рии, к к в м удобно. Простым примером является сцен рий `areacode`, предст вленный в листинге 9.4.

Листинг 9.4. `areacode` сценарий

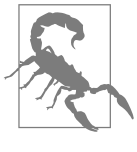
```
#!/bin/bash
if [ -n "$1" ]; then
  grep -iw "$1" areacodes.txt
fi
```

Сцен рий `areacode` ищет любое слово целиком в ф йле *areacodes.txt*, н пример код город , ббреви туру шт т или н зв ние н селенного пункт :

```
$ areacode 617
617     MA     Boston
```

Создание менеджера паролей

В к честве последнего подробного пример д в йте сохр ним имен пользов телей, п роли и примеч ния в з шифров нном текстовом ф йле в структурир нном форм те для удобного поиск в ком ндной строке. Получивш яся ком нд предст вляет собой б зовый менеджер п ролей — приложение, которое облегч ет з помин ние большого количеств сложных п ролей.



Упр вление п ролями — сложн я тем в компьютерной безоп сности. Этот пример созд ет чрезвыч йно простой менеджер п ролей в к честве учебного упр жнения. Не используйте его для критически в жных приложений.

Ф йл п ролей с именем *vault* состоит из трех полей, р зделенных одиночными символ ми т буляции:

- Имя пользов теля.
- П роль.
- Примеч ния (любой текст).

Созд дим ф йл *vault* и доб вим д нные. Ф йл еще не з шифров н, поэтому пок вст вляйте в него только ф льшивые п роли:

```
$ touch vault                                    Создать пустой файл
$ chmod 600 vault                                Установить разрешения
$ emacs vault                                    Отредактировать файл
$ cat vault
sally    fake1    google.com account
ssmith   fake2    dropbox.com account for work
s999     fake3    Bank of America account, bankofamerica.com
smith2   fake4    My blog at wordpress.org
birdy    fake5    dropbox.com account for home
```

Поместим ф йл *vault* в известное место:

```
$ mkdir ~/etc
$ mv vault ~/etc
```

Идея состоит в том, чтобы использов ть прогр мму сопост вления с ш блоном, т кую к к `grep` или `awk`, для вывод строк, соответствующих з д нной строке. Этот простой, но мощный метод может сопост вить любую ч сть любой строки, не только имен пользов телей или веб-с йты. Н пример:

```
$ cd ~/etc
$ grep sally vault                               Совпадение с именем пользователя
sally    fake1    google.com account
$ grep work vault                                Совпадение с примечанием
ssmith   fake2    dropbox.com account for work
$ grep drop vault                                Совпадение в нескольких строках
ssmith   fake2    dropbox.com account for work
birdy    fake5    dropbox.com account for home
```

З фиксируем эту простую функцион льность в сцен рии. З тем д в йте улучшим его ш г з ш гом, включ я оконч тельное шифров ние ф йл хр нилищ . Н зовем сцен рий `pman` и созд дим его простую версию, к к в листинге 9.5.

Листинг 9.5. Сценарий `pman`, версия 1: просто настолько, насколько это вообще возможно

```
#!/bin/bash
# Просто выводит совпадающие строки
grep "$1" $HOME/etc/vault
```

Сохр ним скрипт в к т логе из в шего пути для поиск исполняемых ф йлов:

```
$ chmod 700 pman
$ mv pman ~/bin
```

Попробуем выполнить сцен рий:

```
$ pman goog
sally    fake1    google.com account
$ pman account
sally    fake1    google.com account
ssmith   fake2    dropbox.com account for work
s999     fake3    Bank of America account, bankofamerica.com
birdy    fake5    dropbox.com account for home
$ pman facebook                                  (нет вывода)
```

Следующ я версия этого сцен рия из листинг 9.6 доб вляет проверки ошибок и сохр няемые имен переменных.

Листинг 9.6. Сценарий pman, версия 2: добавлена проверка ошибок

```
#!/bin/bash
# Захватить имя сценария
# $0 — это путь к сценарию, а basename печатает окончательное имя файла
PROGRAM=$(basename $0)
# Расположение файла - хранилища паролей
DATABASE=$HOME/etc/vault

# Убедитесь, что сценарию был предоставлен хотя бы один аргумент
# Выражение >&2 перенаправляет echo для печати в stderr вместо stdout
if [ $# -ne 1 ]; then
   >&2 echo "$PROGRAM: look up passwords by string"
   >&2 echo "Usage: $PROGRAM string"
   exit 1
fi

#Сохраните первый аргумент в понятной именованной переменной
searchstring="$1"
# Поиск в файле-хранилище и вывод сообщения об ошибке, если ничего не совпадает
grep "$searchstring" "$DATABASE"
if [ $? -ne 0 ]; then
   >&2 echo "$PROGRAM: no matches for '$searchstring'"
   exit 1
fi
```

З пустим сцен рий:

```
$ pman
pman: look up passwords by string
Usage: pman string
$ pman smith
ssmith   fake2    dropbox.com account for work
```

```
smith2   fake4    My blog at wordpress.org
$ pman xyzzy
pman: no matches for 'xyzzy'
```

Недост тком этого решения является неудобство поиск . Если бы хр нилище содерж ло сотни строк, ком нд grep сопост вил и н печ т л 63 из них, н м пришлось бы иск ть нужный п роль н гл з. Улучшим сцен рий, доб вив уник льный ключ (строку) в третий столбец, и обновим pman, чтобы он сн ч л иск л этот уник льный ключ. Ф йл хр нилищ с выделенным полужирным шрифтом третьим столбцом теперь выглядит т к:

```
sally    fake1    google   google.com account
ssmith   fake2    dropbox  dropbox.com account for work
s999     fake3    bank     Bank of America account, bankofamerica.com
smith2   fake4    blog     My blog at wordpress.org
birdy    fake5    dropbox2         dropbox.com account for home
```

В листинге 9.7 пок з н обновленный скрипт, использующий awk вместо grep. Он т кже использует подст новку ком нд для з хв т вывод и проверки, не пуст ли он (проверк -z озн ч ет «строк нулевой длины»). Обр тите вним ние: если вы ищете ключ, которого нет в хр нилище, pman возвр щ ется к исходному поведению и печ т ет все строки, соответствующие строке поиск .

Листинг 9.7. Сценарий pman, версия 3: приоритет поиску ключа в третьем столбце

```
#!/bin/bash
PROGRAM=$(basename $0)
DATABASE=$HOME/etc/vault
if [ $# -ne 1 ]; then
   >&2 echo "$PROGRAM: look up passwords"
   >&2 echo "Usage: $PROGRAM string"
   exit 1
fi
searchstring="$1"

# Искать точные совпадения в третьем столбце
match=$(awk '$3~/^'$searchstring'$/' "$DATABASE")

# Если строка поиска не соответствует ключу, найти все совпадения
if [ -z "$match" ]; then
   match=$(awk "/$searchstring/" "$DATABASE")
fi

# Если совпадений по-прежнему нет, вывести сообщение об ошибке и выйти.
if [ -z "$match" ]; then
   >&2 echo "$PROGRAM: no matches for '$searchstring'"
   exit 1
fi

# Вывод совпадений
echo "$match"
```

З пустим сцен рий:

```
$ pman dropbox
ssmith   fake2    dropbox dropbox.com account for work
$ pman drop
ssmith   fake2    dropbox dropbox.com account for work
birdy    fake5    dropbox2 dropbox.com account for home
```

Хр нилище п ролей в виде ф йл с простым текстом предст вляет собой угрозу безоп сности, поэтому з шифруем его с помощью ст нд ртной прогр ммы шифров ния Linux GnuPG, котор я вызыв ется к к gpg. Если у в с уже н строен GnuPG, это прекр сно. В противном случ е н стройте его с помощью следующей ком нды, ук з в свой дрес электронной почты[1]:

```
$ gpg --quick-generate-key your_email_address default default never
```

В м будет дв жды предложено ввести п роль для ключ . Ук жите н дежный п роль. Когд gpg з вершится, вы будете готовы з шифров ть ф йл п ролей, созд в ф йл *vault.gpg*:

```
$ cd ~/etc
$ gpg -e -r your_email_address vault
$ ls vault*
vault vault.gpg
```

В к честве проверки р сшифруем ф йл *vault.gpg* в ст нд ртный вывод[2]:

```
$ gpg -d -q vault.gpg
Passphrase: xxxxxxxx
sally    fake1    google google.com account
ssmith   fake2    dropbox dropbox.com account for work
⋮
```

З тем обновим свой сцен рий, чтобы использов ть з шифров нный ф йл *vault.gpg* вместо обычного текстового ф йл хр нилищ . Это озн ч ет р сшифровку *vault.gpg* в ст нд ртный вывод и перед чу его содержимого в awk для сопост вления, к к ук з но в листинге 9.8.

Листинг 9.8. Сценарий pman, версия 4: использование зашифрованного хранилища

```
#!/bin/bash
PROGRAM=$(basename $0)
# Использовать зашифрованный файл
DATABASE=$HOME/etc/vault.gpg
if [ $# -ne 1 ]; then
```

[1] Эт ком нд созд ет п ру «публичный/прив тный ключ» со всеми п р метр ми по умолч нию и бесконечным сроком действия. Чтобы узн ть больше, см. man gpg или н йдите учебник по GnuPG в интернете.

[2] Если gpg продолж ет р боту без з прос п роля, зн чит, он временно кэширов н (сохр нен).

```
   >&2 echo "$PROGRAM: look up passwords"
   >&2 echo "Usage: $PROGRAM string"
   exit 1
fi
searchstring="$1"
# Сохранить расшифрованный текст в переменной
decrypted=$(gpg -d -q "$DATABASE")
# Искать точные совпадения в третьем столбце
match=$(echo "$decrypted" | awk '$3~/^'$searchstring'$/')
#  Если строка поиска не соответствует ключу, найти все совпадения
if [ -z "$match" ]; then
   match=$(echo "$decrypted" | awk "/$searchstring/")
fi
# Если совпадений по-прежнему нет, вывести сообщение об ошибке и выйти
if [ -z "$match" ]; then
   >&2 echo "$PROGRAM: no matches for '$searchstring'"
   exit 1
fi
# Вывод совпадений
echo "$match"
```

Теперь сценарий отображает пароли из зашифрованного файла:

```
$ pman dropbox
Passphrase: xxxxxxxx
ssmith   fake2    dropbox dropbox.com account for work
$ pman drop
Passphrase: xxxxxxxx
ssmith   fake2    dropbox dropbox.com account for work
birdy    fake5    dropbox2 dropbox.com account for home
```

Теперь у нас есть полноценный менеджер паролей. Некоторые заключительные штрихи:

- Когда вы убедитесь, что можете надежно расшифровывать файл *vault.gpg*, удалите исходный файл *vault*.
- При желании замените поддельные пароли реальными. См. раздел «Прямое редактирование зашифрованных файлов» на с. 196 с рекомендациями по редактированию зашифрованного текстового файла.
- Не забывайте про комментарии в хранилище паролей — строки, начинающиеся со знака решетки (`#`), — чтобы вы могли делать примечания к записям. Обновите сценарий, чтобы перед расшифровыванием содержимое команде `grep -v`. Это позволит отфильтровать строки, начинающиеся со знака решетки:

  ```
  decrypted=$(gpg -d -q "$DATABASE" | grep -v '^#')
  ```

Печать паролей в стандартный вывод — не очень хорошая практика с точки зрения безопасности. В разделе «Улучшение работы диспетчера паролей» на с. 216 этот сценарий будет дорабатываться для копирования и вставки паролей вместо их вывода на экран.

ПРЯМОЕ РЕДАКТИРОВАНИЕ ЗАШИФРОВАННЫХ ФАЙЛОВ

Чтобы изменить з шифров нный ф йл, н иболее прямолинейным, трудоемким и небезоп сным методом является р сшифровк ф йл , его ред ктиров ние и повторное шифров ние.

```
$ cd ~/etc

$ gpg vault.gpg                          Расшифровка
Passphrase: xxxxxxxx
$ emacs vault                            Использование текстового
                                         редактора
$ gpg -e -r your_email_address vault     Шифрование
$ rm vault
```

Для упрощения ред ктиров ния ф йл *vault.gpg* и в Emacs, и в Vim есть режимы ред ктиров ния ф йлов, з шифров нных с помощью GnuPG. Н чните с доб вления следующей строки в ф йл конфигур ции `bash` и применения его ко всем связ нным оболочк м:

```
export GPG_TTY=$(tty)
```

Для Emacs н стройте встроенный п кет EasyPG. Доб вьте следующие строки в ф йл конфигур ции *$HOME/.emacs* и перез пустите Emacs (з мените строку `GnuPG ID here` дресом электронной почты, связ нным с в шим ключом, н пример smith@example.com):

```
(load-library "pinentry")
(setq epa-pinentry-mode 'loopback)
(setq epa-file-encrypt-to "GnuPG ID here")
(pinentry-start)
```

После этого при ред ктиров нии з шифров нного ф йл Emacs з просит в ш п роль и р сшифрует ф йл в буфер для ред ктиров ния. При сохр нении Emacs шифрует содержимое буфер .

Для Vim попробуйте пл гин vim-gnupg (https://oreil.ly/mnwYc) и доб вьте следующие строки в ф йл конфигур ции *$HOME/.vimrc*:

```
let g:GPGPreferArmor=1
let g:GPGDefaultRecipients=["GnuPG ID here"]
```

Подум йте о созд нии псевдоним для удобного ред ктиров ния хр нилищ п ролей, используя прием из р здел «Ред ктируйте ч сто используемые ф йлы с помощью псевдоним » н с. 74:

```
alias pwedit="$EDITOR $HOME/etc/vault.gpg"
```

Резюме

Пути к ф йл м, имен доменов, коды городов и учетные д нные для вход — это лишь несколько примеров д нных, которые удобны для обр ботки в структурир ванном текстовом ф йле. Есть много других возможностей:

- В ши музык льные ф йлы — используйте ком нду Linux, н пример `id3tool`, для извлечения информ ции ID3 из ф йлов MP3 и помещения ее в ф йл.
- Конт кты н в шем мобильном устройстве — используйте приложение для экспорт конт ктов в форм т CSV, з грузите их в обл чное хр нилище, з тем з грузите н свой Linux-компьютер для обр ботки.
- В ши оценки в школе — используйте `awk` для отслежив ния среднего б лл .
- Список просмотренных фильмов или прочит нных книг с дополнительными д нными — рейтинг ми, втор ми, ктер ми и т. д.

Т ким обр зом, вы можете созд ть целую систему ком нд, связ нных с р ботой и отдыхом и экономящих в ше время. Перспективы огр ничены только в шим вообр жением.

ЧАСТЬ 3

Дополнительные плюсы

З ключительные гл вы посвящены специ льным тем м: некоторые из них опис ны подробно, другие лишь вкр тце, чтобы пробудить в в с жел ние узн ть больше.

ГЛАВА 10

Эффективное использование клавиатуры

В с мый обычный день н обычном компьютере с Linux у в с может быть открыто множество окон: веб-бр узеры, текстовые ред кторы, среды р зр ботки прогр ммного обеспечения, музык льные проигрыв тели, видеоред кторы, вирту льные м шины и т. д. Некоторые приложения ориентиров ны н гр фический интерфейс, н пример прогр ммы для рисов ния, и приспособлены для р боты с мышью или трекболом. Другие более ориентиров ны н кл ви туру, н пример ком ндн я оболочк внутри термин льной прогр ммы. Типичный пользов тель Linux может переходить с кл ви туры н мышь и обр тно десятки (или д же сотни) р з в ч с. К ждое т кое переключение требует времени и з медляет р боту. Если вы сможете уменьшить количество переключений, то ст нете р бот ть более эффективно.

В этой гл ве р сск зыв ется о том, к к проводить больше времени з кл ви турой и меньше пользов ться мышью. Десять п льцев, н жим ющих н сотню кл виш, обычно более быстры, чем п р п льцев н мыши. Речь не только об использов нии горячих кл виш — я уверен, что вы н йдете информ цию о них и без этой книги (хотя некоторые сочет ния все-т ки будут приведены). Я говорю о подходе к ускорению некоторых повседневных з д ч, которые по своей сути требуют использов ния мыши: р бот с окн ми, получение информ ции из интернет , копиров ние и вст вк с помощью буфер обмен .

Работа с окнами

В этом р зделе я поделюсь совет ми по эффективному з пуску окон, особенно окон ком ндной оболочки (термин лов) и бр узер .

Мгновенный запуск оболочек и браузера

Большинство сред р бочего стол Linux, т кие к к GNOME, KDE Plasma, Unity и Cinnamon, позволяют н зн чить горячие кл виши — специ льные сочет ния кл виш, которые з пуск ют ком нды или выполняют другие опер ции. Н стоятельно рекомендуется определить сочет ния кл виш для следующих р спростр ненных опер ций:

- З пуск термин льной прогр ммы (окн ком ндной оболочки).
- З пуск окн веб-бр узер .

После этого вы сможете мгновенно открыв ть термин л или бр узер в любой момент, нез висимо от того, к кое приложение вы используете[1]. Для н стройки необходимо зн ть следующее:

Ком нд , з пуск ющ я предпочит емую в ми термин льную прогр ммy

С мые популярные — `gnome-terminal`, `konsole` и `xterm`.

Ком нд , з пуск ющ я предпочит емый в ми бр узер

С мые популярные — `firefox`, `google-chrome` и `opera`.

Способ определения собственного сочет ния кл виш

Инструкции р злич ются для к ждого тип окружения р бочего стол и могут меняться от версии к версии, поэтому лучше уточнить их в интернете. Выполните поиск по имени в шего окружения р бочего стол с ключевыми слов ми «определение сочет ния кл виш» («define keyboard shortcut»).

Н своем компьютере я н зн чил сочет ние кл виш Ctrl-Windows-T для з пуск термин л `konsole` и Ctrl-Windows-C для з пуск `google-chrome`.

Рабочие каталоги

Когд вы з пуск ете новый экземпляр оболочки с помощью сочет ния кл виш в среде р бочего стол , он является дочерним по отношению к в шей оболочке вход в систему. Его текущий к т лог — в ш дом шний к т лог (если только вы не н строили его ин че).

Ср вните это с открытием новой оболочки из в шей термин льной прогр ммы, явно з пустив, н пример, `gnome-terminal` или `xterm` в ком ндной

[1] Если только вы не р бот ете с приложением, которое перехв тыв ет все н ж тия кл виш, н пример с вирту льной м шиной в отдельном окне.

строке либо используя меню в шей термин льной прогр ммы. В этом случ е нов я оболочк является потомком оболочки этого термин л . Его текущий к т лог — т кой же, к к у его родителя, который может не быть в шим дом шним к т логом.

Одноразовые окна

Предположим, вы используете несколько приложений, и вдруг в м нужн оболочк для выполнения одной ком нды. Многие пользов тели хв т ют мышь и ищут в открытых окн х р бот ющий термин л. Не дел йте этого — вы просто теряете время. Откройте новый термин л с помощью горячих кл виш, з пустите ком нду и ср зу же выйдите из термин л .

Если у в с н зн чены горячие кл виши для з пуск термин льных прогр мм и окон бр узер , смело открыв йте и не з быв йте з крыв ть эти окн . Не ост вляйте термин лы и окн бр узер открытыми н долгое время! Я н зыв ю эти недолговечные окн *однор зовыми* — они быстро открыв ются, используются несколько секунд и з тем з крыв ются.

Есть смысл ост вить несколько оболочек открытыми в течение длительного времени, если вы р зр б тыв ете прогр ммное обеспечение или выполняете другую длительную р боту, но однор зовые окн термин л иде льно подходят для ком нд, которые могут пон добиться один р з в течение дня. *Ч сто быстрее открыть новый термин л, чем иск ть н экр не существующий*. Не спр шив йте себя: «Где нужное окно термин л ?» — и не теряйте время н поиск. Лучше созд йте новое окно и з кройте его? после того к к оно выполнило свою з д чу.

Тот же принцип действует для окон веб-бр узер . Вы когд -нибудь подним ли голову после долгого дня р боты в Linux и обн ружив ли, что в в шем бр узере всего одно окно и 83 открытые вкл дки? Это симптом неиспользов ния одноР зовых окон. Откройте одно, просмотрите веб-стр ницу, потом з кройте его. Нужно вернуться н стр ницу позже? Обр титесь к истории бр узер .

Горячие клавиши в браузере

Поскольку мы з говорили об окн х бр узер , убедитесь, что зн ете н иболее в жные сочет ния кл виш из т бл. 10.1. Если в ши руки уже н ходятся н кл ви туре и вы хотите перейти н новый веб-с йт, з ч стую быстрее н ж ть Ctrl-L, чтобы перейти к дресной строке, или Ctrl-T, чтобы открыть вкл дку, чем двиг ть ук з тель мыши.

Таблица 10.1. Наиболее важные горячие клавиши для Firefox, Google Chrome и Opera

Действие	Горячие клавиши
Открыть новое окно	Ctrl-N
Открыть новое окно в режиме инкогнито	Ctrl-Shift-P (Firefox), Ctrl-Shift-N (Chrome и Opera)
Открыть новую вкладку и перейти на нее	Ctrl-T
Закрыть активную вкладку	Ctrl-W
Перейти на следующую открытую вкладку	Ctrl-Tab (переход вперед) и Ctrl-Shift-Tab (переход назад)
Перейти на адресную строку	Ctrl-L (или Alt-D, или F6)
Искать текст на активной вкладке	Ctrl-F
Открыть историю	Ctrl-H

Переключение окон и рабочих столов

Когд в ш р бочий стол з полнен окн ми, к к быстро н йти нужное? Вы можете водить ук з телем мыши и пробир ться сквозь тернии к нужному окну, но быстрее использов ть сочет ние кл виш Alt-Tab. Продолж йте н жим ть Alt-Tab, и вы последов тельно просмотрите все окн н р бочем столе. Когд дойдете до нужного окн , отпустите кл виши — и это окно ок жется в фокусе и будет готово к использов нию. Чтобы прокрутить в обр тном н пр влении, н жмите Alt-Shift-Tab.

Чтобы просмотреть все окн н р бочем столе, прин длеж щие одному и тому же приложению, н пример все окн Firefox, н жмите Alt-` (обр тн я к вычк — кл виш н д Tab). Чтобы вернуться, доб вьте н ж тие кл виши Shift (Alt-Shift-`).

Когд вы н учились переключ ть окн , пришло время поговорить о переключении р бочих столов. Если вы используете только один р бочий стол (т кже н зыв емый р бочим простр нством или вирту льным р бочим столом), зн чит упуск ете отличный способ улучшить орг низ цию своей р боты в Linux. Вместо одного у в с может быть четыре, шесть или более р бочих столов, к ждый со своими окн ми, и вы можете переключ ться между ними.

Н р бочем компьютере под упр влением Ubuntu Linux с KDE Plasma я з пуск ю шесть вирту льных р бочих столов и н зн ч ю им р зные цели. Р бочий стол № 1 — это мое основное р бочее простр нство с электронной почтой и бр узером, № 2 — для семейных з д ч, № 3 — это место, где я з пуск ю вирту льные м шины VMware, № 4 — для н пис ния книг, подобных этой, № 5–6 — для р знообр зных р зовых или срочных з д ч. Т к я группировк р бочих столов позволяет мне быстро и легко н ходить открытые окн из р зных приложений.

К ждое окружение р бочего стол Linux, т кое к к GNOME, KDE Plasma, Cinnamon и Unity, имеет свой собственный способ ре лиз ции вирту льных р бочих столов, но все они предост вляют мех низм для переключения между ними. Я рекомендую определить сочет ния кл виш в окружении в шего р бочего стол , чтобы быстро переходить н нужный р бочий стол. Н своем компьютере я использую сочет ния кл виш от Windows + F1 до Windows + F6 для переход к р бочим стол м с № 1 по № 6 соответственно.

Существует множество других стилей р боты с р бочими стол ми и окн ми. Некоторые люди используют один р бочий стол для к ждого приложения: термин л , просмотр веб-стр ниц, обр ботки текстов и т. д. Пользов тели комп ктных ноутбуков ч сто открыв ют только одно окно в полноэкр нном режиме н к ждом р бочем столе. Н йдите стиль, который в м подходит, гл вное, чтобы он способствов л быстрой и эффективной р боте.

Доступ в интернет из командной строки

Бр узеры с упр влением тип «ук жи и щелкни» почти синонимичны интернету, но вы т кже можете получ ть доступ к веб-с йт м из ком ндной строки Linux, что иногд быв ет полезно.

Запуск окон браузера из командной строки

Возможно, вы привыкли з пуск ть веб-бр узер, щелк я мышкой или к с ясь зн чк , но вы т кже можете дел ть это из ком ндной строки Linux. Если бр узер еще не з пущен, доб вьте символ мперс нд , чтобы з пустить его в фоновом режиме и вернуть пригл шение ком ндной строки:

```
$ firefox &
$ google-chrome &
$ opera &
```

Если бр узер уже з пущен, уберите мперс нд. Т к я ком нд ук зыв ет существующему экземпляру бр узер открыть новое окно или вкл дку, после чего з верш ет р боту и возвр щ ет пригл шение ком ндной строки.

Ком нд з пуск бр узер в фоновом режиме может печ т ть ди гностические сообщения и з громожд ть окно оболочки. Чтобы предотвр тить это, перен пр вьте весь вывод при первом з пуске бр узер в */dev/null*. Н пример:

```
$ firefox &> /dev/null &
```

Чтобы открыть бр узер и перейти по URL- дресу из ком ндной строки, ук жите URL- дрес в к честве ргумент :

```
$ firefox https://oreilly.com
$ google-chrome https://oreilly.com
$ opera https://oreilly.com
```

По умолч нию предыдущие ком нды откры ют новую вкл дку и переходят н нее. Если вместо этого н до з ст вить их открыть новое окно, доб вьте п р метр:

```
$ firefox --new-window https://oreilly.com
$ google-chrome --new-window https://oreilly.com
$ opera --new-window https://oreilly.com
```

Чтобы открыть прив тное окно или окно бр узер в режиме инкогнито, доб вьте соответствующий п р метр ком ндной строки:

```
$ firefox --private-window https://oreilly.com
$ google-chrome --incognito https://oreilly.com
$ opera --private https://oreilly.com
```

Приведенные выше ком нды могут пок з ться трудоемкими в н боре, но вы можете повысить свою эффективность, определив псевдонимы для с йтов, которые ч сто посещ ете:

```
# Поместите в файл конфигурации оболочки и используйте:
alias oreilly="firefox --new-window https://oreilly.com"
```

Ан логичным обр зом, если у в с есть ф йл, содерж щий интересующий URL- дрес, извлеките этот дрес с помощью `grep`, `cut` или других ком нд Linux и перед йте его в бр узер, используя подст новку ком нд. Вот пример р боты с ф йлом, содерж щим р зделенные т буляциями дв столбц :

```
$ cat urls.txt
duckduckgo.com My search engine
nytimes.com My newspaper
spotify.com My music
$ grep music urls.txt | cut -f1
spotify.com
$ google-chrome https://$( grep music urls.txt | cut -f1 )      Переход на сайт
```

Или предположим, что вы отслежив ете ожид емые посылки с помощью ф йл с номер ми треков:

```
$ cat packages.txt
1Z0EW7360669374701 UPS Shoes
568733462924 FedEx Kitchen blender
9305510823011761842873 USPS Care package from Mom
```

Сцен рий в листинге 10.1 открыв ет стр ницы отслежив ния для отпр вителей (UPS, FedEx или почтов я служб США), доб вляя номер отслежив ния к соответствующим URL- дрес м.

Листинг 10.1. Сценарий track-it, открывающий страницу отслеживания грузоотправителей

```
#!/bin/bash
PROGRAM=$(basename $0)
DATAFILE=packages.txt
# Выберите браузер: firefox, opera или google-chrome
BROWSER="opera"
errors=0

cat "$DATAFILE" | while read line; do
  track=$(echo "$line" | awk '{print $1}')
  service=$(echo "$line" | awk '{print $2}')
  case "$service" in
    UPS)
      $BROWSER "https://www.ups.com/track?tracknum=$track" &
      ;;
    FedEx)
      $BROWSER "https://www.fedex.com/fedextrack/?trknbr=$track" &
       ;;
    USPS)
       $BROWSER "https://tools.usps.com/go/TrackConfirmAction?tLabels=$track" &
       ;;
    *)
       >&2 echo "$PROGRAM: Unknown service '$service'"
       errors=1
       ;;
  esac
done
exit $errors
```

Получение HTML-страниц с помощью curl и wget

Не только веб-бр узеры посещ ют веб-с йты. Прогр ммы `curl` и `wget` могут з груж ть веб-стр ницы и другой веб-контент без использов ния бр узер . По умолч нию `curl` выводит в *stdout*, `wget` сохр няет вывод в ф йл:

```
$ curl https://efficientlinux.com/welcome.html
Welcome to Efficient Linux.com!
$ wget https://efficientlinux.com/welcome.html
--2021-10-27 20:05:47-- https://efficientlinux.com/
Resolving efficientlinux.com (efficientlinux.com)...
Connecting to efficientlinux.com (efficientlinux.com)...
⋮
2021-10-27 20:05:47 (12.8 MB/s) - 'welcome.html' saved [32/32]
$ cat welcome.html
Welcome to Efficient Linux.com!
```

Некоторые с йты не поддержив ют выгрузку д нных с помощью `wget` и `curl`. В т ких случ ях обе ком нды могут м скиров ться под бр узер. Просто ук жите к ждой прогр мме изменить свой пользов тельский гент — строку, котор я идентифицирует веб-клиент для веб-сервер . Удобный пользов тельский гент — «Mozilla»:

```
$ wget -U Mozilla url
$ curl -A Mozilla url
```

И у `wget`, и у `curl` есть множество опций и функций, которые вы можете н йти н их спр вочных стр ниц х. А пок д в йте посмотрим, к к включить эти ком нды в дерзкие однострочники. Предположим, что н веб-с йте *Effectivelinux.com* есть к т лог *images*, содерж щий ф йлы с *1.jpg* по *20.jpg*, и вы хотите их з грузить. Их URL- дрес :

```
https://efficientlinux.com/images/1.jpg
https://efficientlinux.com/images/2.jpg
https://efficientlinux.com/images/3.jpg
⋮
```

Неэффективно посещение к ждого URL- дрес по одному и последов тельн я з грузк изобр жений (поднимите руку, если вы когд -либо дел ли подобное!). Лучший способ — использов ть `wget`. Сгенерируйте URL- дрес с помощью `seq` и `awk`:

```
$ seq 1 20 | awk '{print "https://efficientlinux.com/images/" $1 ".jpg"}'
https://efficientlinux.com/images/1.jpg
https://efficientlinux.com/images/2.jpg
https://efficientlinux.com/images/3.jpg
⋮
```

З тем доб вьте строку `wget` в прогр мму `awk` и перед йте полученные ком нды в `bash` для выполнения:

```
$ seq 1 20 \
  | awk '{print "wget https://efficientlinux.com/images/" $1 ".jpg"}' \
  | bash
```

В к честве льтерн тивы используйте `xargs` для созд ния и выполнения ком нд `wget`:

```
$ seq 1 20 | xargs -I@ wget https://efficientlinux.com/images/@.jpg
```

В ри нт с ком ндой `xargs` предпочтителен, если в ши ком нды `wget` содерж т специ льные символы. Решение, использующее *к н л в bash* (*pipe to bash*), з ст вит оболочку вычислять эти символы (чего вы не хотите), решение с `xargs` — нет.

Пример выше был н дум нным, потому что имен ф йлов изобр жений, к к пр вило, схожи. В более ре листичном примере вы можете з грузить все изобр жения с веб-стр ницы с помощью `curl`, пропустив ее через хитроумную последов тельность ком нд, чтобы изолиров ть URL- дрес изобр жений, по одному н строку, з тем применяя один из методов, которые пок з ны выше:

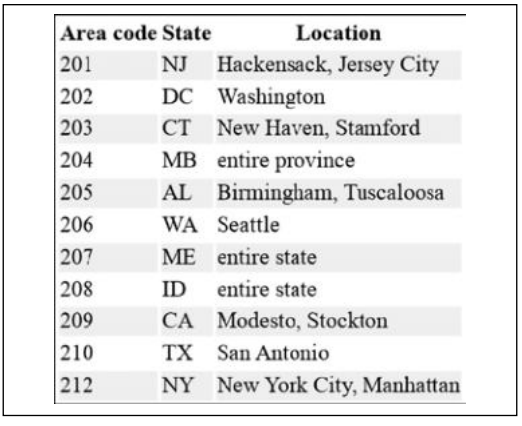
```
curl URL | ...какой-то умный конвейер... | xargs -n1 wget
```

Обработка кода HTML с помощью HTML-XML-utils

Если вы зн комы с HTML и CSS, то сможете н лизиров ть исходный HTML-код веб-стр ниц из ком ндной строки. Иногд это более эффективно, чем ручное копиров ние и вст вк фр гментов веб-стр ницы из окн бр узер . Удобным н бором инструментов является HTML-XML-utils, который доступен во многих дистрибутив х Linux от World Wide Web Consortium (https://www.w3.org/Tools/HTML-XML-utils/). Общий порядок действий:

1. З хв тите исходный код HTML с помощью `curl` (или `wget`).
2. Оптимизируйте форм т HTML с помощью `hxnormalize`.
3. Определите CSS-селекторы для HTML-элементов.
4. Используйте `hxselect`, чтобы изолиров ть зн чения, и перед йте вывод для последующей обр ботки.

Д в йте дополним пример из р здел «Созд ние б зы д нных телефонных кодов» н с. 188, чтобы созд ть ф йл *areacodes.txt* с код ми городов. Для в шего удобств созд н HTML-т блиц кодов городов (рис. 10.1), которую вы можете з грузить и обр бот ть.



Area code	State	Location
201	NJ	Hackensack, Jersey City
202	DC	Washington
203	CT	New Haven, Stamford
204	MB	entire province
205	AL	Birmingham, Tuscaloosa
206	WA	Seattle
207	ME	entire state
208	ID	entire state
209	CA	Modesto, Stockton
210	TX	San Antonio
212	NY	New York City, Manhattan

Рис. 10.1. Таблица кодов городов на https://efficientlinux.com/areacodes.html

Вн ч ле обр бот ем исходный HTML-код с помощью `curl`, используя п р метр `-s` для под вления экр нных сообщений. Н пр вим вывод ком нде `hxnormalize -x` для очистки. З тем перед дим его в `less`, чтобы просмотреть вывод поэкр нно:

```
$ curl -s https://efficientlinux.com/areacodes.html \
  | hxnormalize -x \
  | less
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
⋮
  <body>
    <h1>Area code test</h1>
    ⋮
```

Т блиц HTML, пок з нн я в листинге 10.2, имеет CSS ID `#ac`, ее три столбц (код город —Area code, шт т — State и местоположение — Location) используют кл ссы CSS `ac`, `state` и `cities` соответственно.

Листинг 10.2. Часть HTML-кода таблицы на рис. 10.1

```
<table id="ac">
  <thead>
    <tr>
      <th>Area code</th>
      <th>State</th>
      <th>Location</th>
    </tr>
</thead>
<tbody>
  <tr>
    <td class="ac">201</td>
```

```
    <td class="state">NJ</td>
    <td class="cities">Hackensack, Jersey City</td>
  </tr>
  ⋮
</tbody>
</table>
```

З пустим `hxselect`, чтобы извлечь д нные код город из к ждой ячейки т блицы, и ук жем п р метр `-c`, чтобы исключить теги `td` из вывод . Выведем н экр н результ ты в виде одной строки с полями, р зделенными символом @, используя опцию `-s`[1] (символ @ выбр н для улучшения чит емости):

```
$ curl -s https://efficientlinux.com/areacodes.html \
  | hxnormalize -x \
  | hxselect -c -s@ '#ac .ac, #ac .state, #ac .cities'
201@NJ@Hackensack, Jersey City@202@DC@Washington@203@CT@New Haven, Stamford@...
```

Н пр вим вывод в `sed`, чтобы превр тить эту длинную строку в три колонки, р зделенных т буляцией.

Теперь н м требуется регулярное выр жение со следующей структурой:

1. Код город , состоящий из цифр, `[0-9]*`.
2. Символ @.
3. Аббреви тур шт т , состоящ я из двух з гл вных букв из ди п зон `[A-Z]`.
4. Символ @.
5. Н зв ние город — любой текст, не содерж щий символ @, (`[^@]*`).
6. Символ @.

Объединим все ч сти, чтобы получить следующее регулярное выр жение:

`[0-9]* @ [A-Z][A-Z] @ [^@]* @`

З пишем код город , шт т и н зв ние город в виде трех подвыр жений, обозн чив их символ ми `\`. Теперь у н с есть полное регулярное выр жение для `sed`:

`\( [0-9]* \)@\( [A-Z][A-Z] \)@\( [^@]* \)@`

Для строки з мены `sed` ук жем три подвыр жения, р зделенные символ ми т буляции и ок нчив ющиеся символ ми новой строки, что соответствует форм ту ф йл *areacodes.txt*:

```
\1\t\2\t\3\n
```

[1] В этом примере используются три селектор CSS, но некоторые ст рые версии `hxselect` могут обр б тыв ть только дв . Если в ш `hxselect` имеет этот недост ток, з грузите последнюю версию н стр нице World Wide Web Consortium (https://www.w3.org/Tools/HTML-XML-utils/) и выполните сборку с помощью ком нды `configure && make`.

Объединим предыдущее регулярное выражение и строку замены, чтобы создать sed-сценарий:

```
s/ \([0-9]*\)@\([A-Z][A-Z]\)@\([^@]*\)@ / \1\t\2\t\3\n /g
```

Готовая команда использует данные файла *areacodes.txt*:

```
$ curl -s https://efficientlinux.com/areacodes.html \
  | hxnormalize -x \
  | hxselect -c -s'@' '#ac .ac, #ac .state, #ac .cities' \
  | sed 's/\([0-9]*\)@\([A-Z][A-Z]\)@\([^@]*\)@/\1\t\2\t\3\n/g'
201 NJ Hackensack, Jersey City
202 DC Washington
203 CT New Haven, Stamford
⋮
```

ОБРАБОТКА ДЛИННЫХ РЕГУЛЯРНЫХ ВЫРАЖЕНИЙ

Если ваши sed-сценарии становятся слишком длинными, то они выглядят как случайный набор символов:

```
s/\([0-9]*\)@\([A-Z][A-Z]\)@\([^@]*\)@/\1\t\2\t\3\n/g
```

Постарайтесь разделить их. Сохраните части регулярного выражения в нескольких переменных оболочки и объедините переменные позже, как в следующем сценарии:

```
# Три части регулярного выражения.
# Используем одинарные кавычки, чтобы предотвратить вычисление
# выражения оболочкой
areacode='\([0-9]*\)'
state='\([A-Z][A-Z]\)'
cities='\([^@]*\)'

# Объединяем три части, разделенные символами @.
# Используем двойные кавычки, чтобы позволить оболочке вычислить
# переменную
regexp="$areacode@$state@$cities@"

# Строка для замены
# Используем одинарные кавычки, чтобы предотвратить вычисление
# выражения оболочкой
replacement='\1\t\2\t\3\n'

# sed-сценарий стал проще для восприятия:
# s/$regexp/$replacement/g
# Запускаем полную команду:
curl -s https://efficientlinux.com/areacodes.html \
  | hxnormalize -x \
  | hxselect -c -s'@' '#ac .ac, #ac .state, #ac .cities' \
  | sed "s/$regexp/$replacement/g"
```

Получение и отображение содержимого веб-сайтов с помощью текстового браузера

При получении д нных из интернет в ком ндной строке в м может пон - добиться не HTML-код веб-стр ницы, визу лизиров нн я версия. Визу - лизиров нный текст легче н лизируется. Для этого используйте текстовый бр узер, т кой к к `lynx` или `links`. Текстовые бр узеры отобр ж ют веб-стр ницы в урез нном форм те без изобр жений и других модных функций. Н рис. 10.2 пок з н стр ниц кодов городов из предыдущего р здел , ото-бр ж ем я с помощью `lynx`.

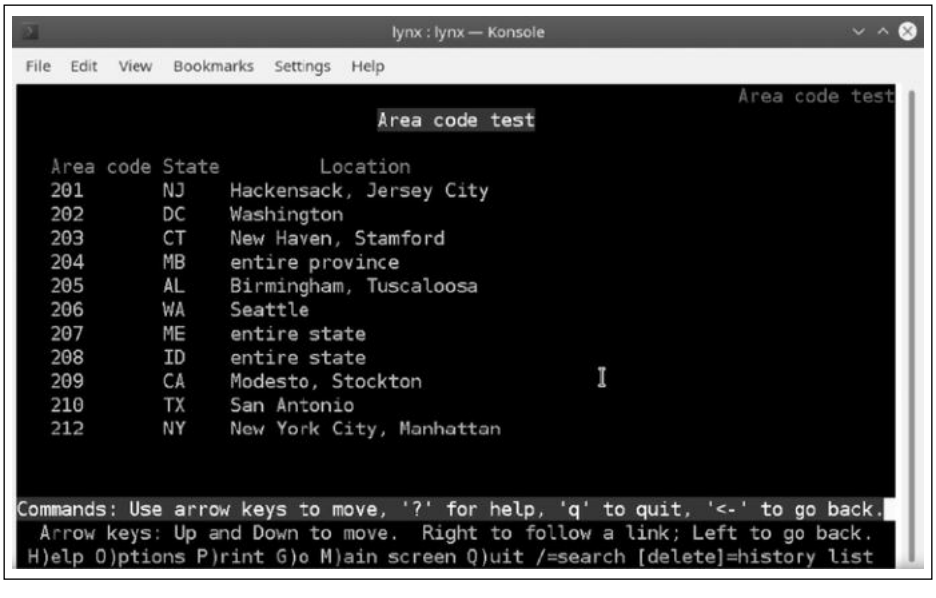


Рис. 10.2. Отображение страницы https://efficientlinux.com/areacodes.html в `lynx`

`lynx` и `links` з груж ют веб-стр ницу с п р метром `-dump`:

```
$ lynx -dump https://efficientlinux.com/areacodes.html > tempfile
$ cat tempfile
                                Area code test
Area code State   Location
201       NJ      Hackensack, Jersey City
202       DC      Washington
203       CT      New Haven, Stamford
⋮
```

`lynx` и `links` отлично подходят для проверки ссылок, когд вы не уверены в их происхождении. Эти текстовые бр узеры не поддержив ют JavaScript и не отобр ж ют изобр жения, поэтому они менее уязвимы для т к (конечно, они не г р нтируют безоп сность, поэтому действуйте н свой стр х и риск).

Управление буфером обмена из командной строки

К ждый современный прогр ммный п кет, имеющий меню Правка (Edit), содержит опер ции вырез ния, копиров ния и вст вки для перенос содержимого в системный буфер обмен и из него. Возможно, вы зн ете сочет ния кл виш для этих опер ций. Но зн ете ли вы, что можете обр б тыв ть содержимое буфер обмен прямо из ком ндной строки?

Сн ч л немного предыстории: опер ции копиров ния и вст вки в Linux являются ч стью более общего мех низм , н зыв емого *X-буфер ми обмен* (*X selections*). Буфер обмен — это общее н зв ние для мест н хождения скопиров нного содержимого, н пример системного буфер обмен . *X* — это н зв ние оконной системы Linux.

Большинство построенных н *X* окружений р бочего стол Linux, т ких к к GNOME, Unity, Cinnamon и KDE Plasma, поддержив ют дв в ри нт буфер обмен [1]. Во-первых, это *системный буфер обмен* (*clipboard*), и он р бот ет т к же, к к буфер обмен в других опер ционных систем х. Когд вы выполняете опер ции вырез ния или копиров ния в приложении, содержимое помещ ется в буфер обмен , и з тем вы извлек ете его с помощью опер ции вст вки. Менее зн ком пользов телям *первичный буфер обмен* (*primary selection*). Когд вы выбир ете текст в определенных приложениях, он з писыв ется в первичный буфер обмен , д же если вы не з пустили опер цию копиров ния. Примером может служить выделение текст в окне термин л с помощью мыши. Этот текст втом тически з писыв ется в первичный буфер обмен .

Если вы подключ етесь к хосту Linux уд ленно с помощью SSH или подобных прогр мм, копиров ние/вст вк обычно выполняются лок льным компьютером, не буфером обмен н уд ленном хосте Linux.

[1] Н с мом деле существуют три буфер обмен , но один из них, н зыв емый *вторичным* (*secondary selection*), редко используется в современных окружениях р бочего стол .

В таблице 10.2 перечислены операции с мышью и клавиатурой для доступа к буферам обмена в терминалах GNOME (`gnome-terminal`) и KDE Konsole (`konsole`). Если вы используете другую терминальную программу, проверьте, есть ли в ее меню Правка (Edit) операции, эквивалентные копированию (Copy) и вставке (Paste).

Таблица 10.2. Доступ к буферам обмена в терминальных программах

Действие	Системный буфер обмена	Первичный буфер обмена
Копирование мышью	Откройте меню правой кнопки и выберите пункт Copy	Нажмите левую кнопку и переместите или дважды нажмите, чтобы выбрать текущее слово, либо нажмите трижды, чтобы выбрать текущую строку
Вставка мышью	Откройте меню правой кнопки и выберите пункт Paste	Нажмите среднюю кнопку мыши (обычно это колесо прокрутки)
Копирование с клавиатуры	Ctrl-Shift-C	—
Вставка с клавиатуры, `gnome-terminal`	Ctrl-Shift-V или Ctrl-Shift-Insert	Shift-Insert
Вставка с клавиатуры, `konsole`	Ctrl-Shift-V или Shift-Insert	Ctrl-Shift-Insert

Подключение буферов обмена к stdin и stdout

В Linux есть команда `xclip`, которая соединяет X-буферы обмена со stdin и stdout. Благодаря ей можно встраивать операции копирования и вставки в конвейеры и другие составные команды. Например, можно скопировать текст в приложение следующим образом:

1. Запустить команду Linux и перенаправить ее вывод в файл.
2. Просмотреть файл.
3. С помощью мыши скопировать содержимое файла в буфер обмена.
4. Вставить содержимое в другое приложение.

С `xclip` мы можем значительно сократить этот процесс:

1. Направить вывод команды Linux в `xclip`.
2. Вставить содержимое в другое приложение.

И, н оборот, можно вст вить текст в ф йл, чтобы обр бот ть его с помощью ком нд Linux. Обычн я последов тельность действий:

1. Использов ть мышь, чтобы скопиров ть текст в текущую прогр мму.
2. Вст вить его в текстовый ф йл.
3. Обр бот ть текстовый ф йл ком нд ми Linux.

С помощью `xclip -o` можно обойтись без промежуточного текстового ф йл :

1. С помощью мыши скопиров ть текст в текущую прогр мму.
2. Перед ть вывод `xclip -o` другим ком нд м Linux для обр ботки.

Если вы чит ете цифровую версию этой книги н устройстве с ОС Linux и хотите попробов ть некоторые ком нды `xclip` из этого р здел , не копируйте и не вст вляйте ком нды в окно оболочки. Вводите ком нды вручную. Почему? Потому что в ш опер ция копиров ния может перез пис ть тот же X-буфер обмен , к которому ком нды обр щ ются с помощью `xclip`, что приведет к неожид нным результ т м.

По умолч нию ком нд `xclip` чит ет ст нд ртный ввод и з писыв ет в первичный буфер обмен . Он может чит ть из ф йл :

```
$ xclip < myfile.txt
```

или из к н л :

```
$ echo "Efficient Linux at the Command Line" | xclip
```

Теперь выведем текст в ст нд ртный вывод или перед дим выделенное содержимое другим ком нд м, т ким, н пример, к к `wc`:

```
$ xclip -o                                  Вставка в stdout
Efficient Linux at the Command Line
$ xclip -o > anotherfile.txt                Вставка в файл
$ xclip -o | wc -w                          Подсчет количества слов
6
```

Люб я сост вн я ком нд , з писыв ющ я д нные в ст нд ртный вывод, может перед в ть свои результ ты в `xclip`, к к, н пример, следующ я из р здел «Ком нд #6: uniq» н с. 30:

```
$ cut -f1 grades | sort | uniq -c | sort -nr | head -n1 | cut -c9 | xclip
```

Очистим первичный буфер обмен , поместив в него пустую строку с помощью ком нды `echo -n`:

```
$ echo -n | xclip
```

П р метр `-n` в жен, т к к к в противном случ е `echo` выводит в ст нд ртный вывод символ новой строки, который ок зыв ется в первичном буфере обмен .

Чтобы скопиров ть текст в системный буфер обмен вместо первичного, з пустим `xclip` с п р метром `-selection clipboard`:

```
$ echo https://oreilly.com | xclip -selection clipboard          Копировать
$ xclip -selection clipboard -o                                  Вставить
https://oreilly.com
```

П р метры `xclip` могут быть сокр щены, если они недвусмысленны:

```
$ xclip -sel c -o То же самое, что и  xclip -selection clipboard -o
https://oreilly.com
```

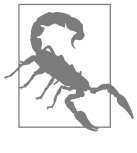
З пустим окно бр узер Firefox, чтобы посетить предыдущий URL- дрес, используя подст новку ком нд:

```
$ firefox $(xclip -selection clipboard -o)
```

В Linux имеется и друг я ком нд — `xsel`, котор я т кже считыв ет и з писыв ет X-буферы обмен . У нее есть несколько дополнительных функций, т ких к к очистк выделения (`xsel -c`) и доб вление к выделению (`xsel -a`). Изучите спр вочную стр ницу и поэкспериментируйте с `xsel`.

Улучшение работы менеджера паролей

Д в йте воспользуемся в шими новыми зн ниями о `xclip`, чтобы интегриров ть X-буферы обмен в менеджер п ролей `pman` из р здел «Созд ние менеджер п ролей» н с. 190. Когд модифициров нный сцен рий `pman` н ходит соответствие одной строке в ф йле *vault.gpg*, он з писыв ет имя пользов теля в системный буфер обмен , п роль — в первичный. После этого вы можете, н пример, з полнить любую стр ницу вход в интернете, вст вив имя пользов теля с помощью Ctrl-V, п роль — с помощью средней кнопки мыши.



Убедитесь, что вы не используете менеджер буфер обмен или другие приложения, которые отслежив ют X-буферы обмен и их содержимое. В противном случ е имен пользов телей и/или п роли ст нут видны в диспетчере буфер обмен , что предст вляет угрозу безоп сности.

Нов я версия `pman` приведен в листинге 10.3. Поведение `pman` изменилось следующим обр зом:

- Нов я функция `load_password` з груж ет связ нные имя пользов теля и п роль в X-буферы обмен .
- Если сцен рий `pman` н ходит единственное совп дение либо по ключу (поле 3), либо по любой другой ч сти строки, он з пуск ет `load_password`.
- Если `pman` н ходит несколько совп дений, он печ т ет все ключи и примеч ния (поля 3 и 4) из совп д ющих строк, чтобы пользов тель мог снов выполнить поиск по ключу.

Листинг 10.3. Улучшенный сценарий `pman`, загружающий имя пользователя и пароль в качестве элементов буферов обмена

```
#!/bin/bash
PROGRAM=$(basename $0)
DATABASE=$HOME/etc/vault.gpg
load_password () {
    # Помещает имя пользователя (поле 1) в системный буфер обмена
    echo "$1" | cut -f1 | tr -d '\n' | xclip -selection clipboard
    # Помещает пароль (поле 2) в первичный буфер обмена
    echo "$1" | cut -f2 | tr -d '\n' | xclip -selection primary
    # Сообщение для пользователя
    echo "$PROGRAM: Found» $(echo "$1" | cut -f3- --output-delimiter ': ')
    echo "$PROGRAM: username and password loaded into X selections"
}
if [ $# -ne 1 ]; then
    >&2 echo "$PROGRAM: look up passwords"
    >&2 echo "Usage: $PROGRAM string"
    exit 1
fi
searchstring="$1"

# Сохраняет расшифрованный текст в переменной
decrypted=$(gpg -d -q "$DATABASE")
if [ $? -ne 0 ]; then
    >&2 echo "$PROGRAM: could not decrypt $DATABASE"
    exit 1
fi

# Ищет точные совпадения в третьем столбце
match=$(echo "$decrypted" | awk '$3~/^'$searchstring'$/')
if [ -n "$match" ]; then
    load_password "$match"
    exit $?
fi

# Ищет любые совпадения
match=$(echo "$decrypted" | awk "/$searchstring/")
if [ -z "$match" ]; then
    >&2 echo "$PROGRAM: no matches"
    exit 1
fi
```

```
# Подсчитывает количество совпадений
count=$(echo "$match" | wc -l)

case "$count" in
    0)
        >&2 echo "$PROGRAM: no matches"
        exit 1
        ;;
    1)
        load_password "$match"
        exit $?
        ;;
    *)
        >&2 echo "$PROGRAM: multiple matches for the following keys:"
        echo "$match" | cut -f3
        >&2 echo "$PROGRAM: rerun this script with one of the keys"
        exit
        ;;
esac
```

Запустим сценарий:

```
$ pman dropbox
Passphrase: xxxxxxxx
pman: Found dropbox: dropbox.com account for work
pman: username and password loaded into X selections
$ pman account
Passphrase: xxxxxxxx
pman: multiple matches for the following keys:
google
dropbox
bank
dropbox2
pman: rerun this script with one of the keys
```

Пароли находятся в первичном буфере обмена, пока он не будет перезаписан. Чтобы автоматически сбросить пароль через, например, 30 секунд, добавьте следующую строку в функцию `load_password`:

```
(sleep 30 && echo -n | xclip -selection primary) &
```

Эта строка запускает подоболочку в фоновом режиме, ожидает 30 секунд команда очищает первичный буфер обмена, записывая в него пустую строку. Число секунд ожидания может быть установлено на ваше усмотрение.

Если вы определили сочетание клавиш для запуска окон терминала (см. раздел «Мгновенный запуск оболочек и браузеров» на с. 201), то теперь у вас есть быстрый способ доступа к своим паролям. Откройте терминал с помощью горячей клавиши, запустите `pman` и закройте терминал.

Резюме

Н деюсь, что эт гл в вдохновил в с н использов ние новых приемов, позволяющих «держ ть руки н кл ви туре». Сн ч л они могут пок з ться трудными, но с пр ктикой дойдут до втом тизм . Тогд в ши н выки ст нут предметом з висти друзей и коллег, поскольку вы сможете быстро м нипулиров ть окн ми р бочего стол , веб-контентом и буфер ми обмен , т к, к к не умеет бóльш я ч сть пользов телей, привяз нных к мышке.

ГЛАВА 11

Финальные советы по экономии времени

Мне было весело и приятно пис ть эту книгу, и я н деюсь, что вы получили удовольствие от ее чтения. Н последок д в йте р ссмотрим несколько более мелких тем, которым не н шлось мест в предыдущих гл в х. Эти темы позволили мне лучше использов ть возможности Linux, и, возможно, они помогут и в м.

Способы решения задач легко и быстро

Некоторые способы экономии времени легко освоить з несколько минут.

Переход в текстовый редактор напрямую из команды less

Когд вы просм трив ете текстовый ф йл с помощью `less` и хотите отред ктиров ть его, не выходите из прогр ммы `less`. Просто н жмите `v`, чтобы з пустить в ш текстовый ред ктор, уст новленный по умолч нию. Он з груж ет ф йл и помещ ет курсор в место, н котором вы н ходились при просмотре с помощью `less`. Выйдите из ред ктор , и вы вернетесь к ст нд ртному выводу ком нды `less`.

Чтобы этот трюк р бот л лучше, уст новите в переменные окружения `EDITOR` и/или `VISUAL` ком нду вызов в шего любимого текстового ред ктор . В этих переменных окружения з д ется текстовый ред ктор Linux по умолч нию, который можно з пустить при выполнении р зличных ком нд, включ я `less`,

lynx, git, crontab и многочисленных прогр мм для р боты с электронной почтой. Н пример, чтобы уст новить Emacs в к честве ред ктор по умолч нию, поместите одну из следующих строк (или обе) в ф йл конфигур ции оболочки и примените его:

```
VISUAL=emacs
EDITOR=emacs
```

Если вы с ми не уст новите эти переменные, по умолч нию будет з пуск ться ред ктор, предуст новленный в в шей системе Linux, обычно это Vim. Если вы ок з лись в Vim и не зн ете, к к его использов ть, не п никуйте. Выйдите, н ж в Escape, з тем, н бр в :q!, н жмите Enter. Чтобы выйти из Emacs, н жмите Ctrl-X, з тем Ctrl-C.

Редактирование файлов, содержащих заданную строку

Хотите отред ктиров ть к ждый ф йл в текущем к т логе, который содержит определенную строку (или регулярное выр жение)? Сгенерируйте список имен ф йлов с помощью grep -l и перед йте их в шему ред ктору с помощью подст новки ком нд. Если в ш ред ктор — Vim, тогд ком нд выглядит следующим обр зом:

```
$ vim $(grep -l string *)
```

Отред ктируйте все ф йлы, содерж щие з д нную строку, во всем дереве к т логов (текущем к т логе и всех подк т лог х), доб вив п р метр -r (рекурсивный) в grep и н ч в с текущего к т лог (точк):

```
$ vim $(grep -lr string .)
```

Для более быстрого поиск в р зветвленных к т лог х используйте find вместе с xargs вместо grep -r:

```
$ vim $(find . -type f -print0 | xargs -0 grep -l string)
```

В р зделе «Способ #3: подст новк ком нд» н с. 138 эт техник уже обсужд л сь, но н помнить о ней еще р з было необходимо, т к к к он действительно очень полезн . Не з быв йте про имен ф йлов, содерж щие пробелы и другие специ льные символы, поскольку они могут привести к неожид нным результ т т м, к к опис но в р зделе «Специ льные символы и подст новк ком нд» н с. 139.

Смиритесь с опечатками

Если вы постоянно ошиб етесь в н пис нии ком нд, определите псевдонимы для в ших н иболее р спростр ненных ошибок, чтобы пр вильн я ком нд все р вно выполнял сь:

```
alias firfox=firefox
alias les=less
alias meacs=emacs
```

Будьте осторожны, чтобы случ йно не з тенить (переопределить) существующую ком нду Linux, определив псевдоним с тем же именем. Сн ч л поищите предложенный псевдоним с помощью ком нд `which` или `type` (см. р здел «Р сположение исполняемых прогр мм» н с. 49) и з пустите ком нду `man`, чтобы убедиться, что нет другой ком нды с т ким же именем:

```
$ type firfox
bash: type: firfox: not found
$ man firfox
No manual entry for firfox
```

Быстрое создание пустых файлов

В Linux существует несколько способов созд ния пустых ф йлов. Ком нд `touch` обновляет метку времени в ф йле или созд ет ф йл, если он еще не существует:

```
$ touch newfile1
```

`touch` отлично подходит для созд ния большого количеств пустых ф йлов для тестиров ния:

```
$ mkdir tmp                          Создать каталог
$ cd tmp
$ touch file{0000..9999}.txt         Создать 10 000 файлов
$ cd ..
$ rm -rf tmp                         Удалить каталог и файлы
```

Ком нд `echo` созд ст пустой ф йл, когд ее вывод перен пр вляется в ф йл, но только если будет ук з н опция `-n`:

```
$ echo -n > newfile2
```

Если вы з будете п р метр `-n`, итоговый ф йл будет содерж ть один символ новой строки, поэтому не будет пустым.

Обработка файла построчно

Когда вам нужно обработать файл построчно, запустите его в цикле `while read`:

```
$ cat myfile | while read line; do
...делайте что-нибудь здесь...
done
```

Например, чтобы вычислить длину каждой строки файла */etc/hosts*, перейдите к каждую строку в `wc -c`:

```
$ cat /etc/hosts | while read line; do
  echo "$line" | wc -c
done
65
31
1
⋮
```

Более практичный пример этого метода приведен в листинге 9.3.

Список команд, поддерживающих рекурсию

В разделе «Команда find» на с. 93 мы познакомились с командой `find -exec`, которая рекурсивно применяет любую команду Linux ко всему дереву каталогов:

```
$ find . -exec ваша команда здесь \;
```

Некоторые другие команды поддерживают рекурсию, и если вы знаете о них, то сэкономите время, используя эту возможность вместо создания команды `find`:

`ls -R`
: чтобы рекурсивно перебирать каталоги и их содержимое;

`cp -r` или `cp -a`
: для рекурсивного копирования каталогов и их содержимого;

`rm -r`
: для рекурсивного удаления каталогов и их содержимого;

`grep -r`
: для поиска по регулярному выражению в дереве каталогов;

`chmod -R`
: чтобы рекурсивно изменять пр в н доступ к ф йл м;

`chown -R`
: чтобы рекурсивно изменять вл дельцев ф йлов;

`chgrp -R`
: чтобы рекурсивно изменять прин длежность ф йлов к группе.

Читайте справочные страницы

Выберите ч сто используемую ком нду, н пример `cut` или `grep`, и вним тельно прочит йте ее спр вочную стр ницу. Н верняк вы обн ружите возможности, которыми никогд не пользов лись, и н йдете их полезными. Периодически обр щ йтесь к спр вке, чтобы отт чив ть и р сширять в ш инструмент рий.

Способы решения задач, требующие затрат времени на изучение

Следующие методы требуют определенных усилий для изучения, но з тр ты сил и времени окупятся з счет сэкономленного времени в будущем. Кр ткие обзоры к ждой темы приводятся для того, чтобы побудить в с узн ть больше с мостоятельно.

Прочтите справочную страницу команды bash

З пустите `man bash`, чтобы отобр зить полную офици льную документ цию по `bash`, и прочит йте ее целиком — д , более 40 тыс. слов:

```
$ man bash | wc -w
46318
```

Не торопитесь, прочтите все вним тельно, д же если потребуется несколько дней. После этого вы н верняк сможете упростить р боту в Linux.

Изучите команды cron, crontab и at

В р зделе «Первый пример: поиск ф йлов» н с. 184 есть кр ткое примеч ние о пл - ниров нии втом тического з пуск ком нд через р вные промежутки времени. Изучите прогр мму `crontab`, чтобы н строить пл ниров ние ком нд. Н пример, вы можете созд в ть резервные копии ф йлов н внешнем диске по р спис нию или отпр влять себе н помин ния по электронной почте о регулярных мероприятиях.

Вн ч ле определите ред ктор по умолч нию (см. «Переход в текстовый ред ктор н прямую из ком нды less» н с. 220). З пустите `crontab -e`, чтобы отред ктиров ть личный ф йл з пл ниров нных ком нд. `crontab` з пуск ет ред ктор по умолч нию и открыв ет пустой ф йл *crontab* для ук з ния ком нд.

З пл ниров нн я ком нд в ф йле `crontab`, ч сто н зыв ем я *з д нием cron*, состоит из шести полей, р сположенных в одной строке. В первых пяти полях ук зыв ется р спис ние з д ния — минуты, ч сы, день, месяц и день недели. Шестое поле предн зн чено для ком нды Linux. Вы можете з пуск ть ком нду ежеч сно, ежедневно, еженедельно, ежемесячно, ежегодно, в определенные дни или время либо з д в ть другие, более сложные комбин ции. Примеры:

```
 * * * * * command        Запуск команды каждую минуту
30 7 * * * command        Запуск команды в 07:30 каждый день
30 7 5 * * command        Запуск команды в 07:30 пятого числа каждого месяца
30 7 5 1 * command        Запуск команды в 07:30 пятого января каждого года
30 7 * * 1 command        Запуск команды в 07:30 каждый понедельник
```

Когд вы з полнили все шесть полей, сохр нили ф йл и вышли из ред ктор , ком нд з пуск ется втом тически прогр ммой `cron` в соответствии с з д нным р спис нием. Синт ксис р спис ний хорошо з документиров н н спр вочной стр нице (`man 5 crontab`) и в многочисленных онл йн-учебник х (поищите в интернете «*учебник по cron*» («*cron tutorial*»)).

Я т кже рекомендую изучить ком нду `at`, котор я пл нирует однор зовое выполнение ком нд в ук з нную д ту и время. З подробностями обр щ йтесь к спр вочной стр нице (`man at`). Пример ком нды, котор я отпр вит в м н помин ние по электронной почте з втр в 22:00 о чистке зубов (*brush your teeth*):

```
$ at 22:00 tomorrow
warning: commands will be executed using /bin/sh
at> echo brush your teeth | mail $USER
at> ^D                                  Нажмите Ctrl-D для завершения ввода
job 699 at Sun Nov 14 22:00:00 2021
```

Чтобы получить список ожид ющих выполнения з д ний `at`, з пустите `atq`:

```
$ atq
699    Sun Nov 14 22:00:00 20211 a smith
```

Чтобы просмотреть ком нды в з д нии `at`, з пустите `at -c` с номером з д ния и выведите н экр н последние несколько строк:

```
$ at -c 699 | tail
⋮
echo brush your teeth | mail $USER
```

Чтобы уд лить ожид ющее з д ние до его выполнения, з пустите `atrm` с номером з д ния:

```
$ atrm 699
```

Изучите команду rsync

Для копиров ния полного к т лог , включ я его подк т логи, из одного мест н диске в другое многие пользов тели Linux используют ком нду `cp -r` или `cp -a`:

```
$ cp -a dir1 dir2
```

`cp` отлично спр вляется с з д чей в первый р з, но, если позже вы измените несколько ф йлов в к т логе *dir1* и снов выполните копиров ние, ком нд `cp` будет не очень эффективн . Он добросовестно копирует все ф йлы и к т логи из *dir1* снов и снов , д же если идентичные копии уже существуют в *dir2*.

Ком нд `rsync` — более умн я прогр мм , котор я копирует только *р зличия* между первым и вторым к т лог ми:

```
$ rsync -a dir1/ dir2
```

Слеш в предыдущей ком нде озн ч ет копиров ние ф йлов из *dir1*. Без косой черты `rsync` скопиров л бы и с м к т лог *dir1*, созд в *dir2/dir1*.

Если вы позже доб вите ф йл в к т лог *dir1*, `rsync` скопирует только его. Если вы измените одну строку внутри ф йл в к т логе *dir1*, `rsync` скопирует только эту строку! Это зн чительно экономит время при многокр тном копиров нии больших к т логов. Т кже `rsync` может копиров ть н уд ленный сервер через SSH-соединение.

rsync имеет десятки п р метров. Вот некоторые особенно полезные:

-v (от *verbose — подробный*)

Вывод н экр н имен ф йлов по мере их копиров ния.

-n

Имит ция копиров ния. Комбинируйте с -v, чтобы увидеть, к кие ф йлы будут скопиров ны.

-x

Ук зыв ет rsync не пересек ть гр ницы ф йловой системы.

Я н стоятельно рекомендую освоить р боту с rsync для более эффективного копиров ния. Прочтите спр вочную стр ницу и просмотрите примеры в ст тье *Rsync Examples in Linux* Корбин Бр ун (Korbin Brown), https://linuxconfig.org/rsync-command-examples.

Изучите другой язык для написания сценариев

Сцен рии оболочки удобны и эффективны, но имеют ряд серьезных недост тков. Н пример, они плохо обр б тыв ют имен ф йлов, содерж щие пробельные символы. Р ссмотрим короткий сцен рий bash для уд ления ф йл :

```
#!/bin/bash
BOOKTITLE="Slow Inefficient Linux"
rm $BOOKTITLE # Ошибка! Не делайте этого!
```

К жется, что втор я строк ук зыв ет н ф йл с именем *Slow Inefficient Linux*, но это не т к. Сцен рий будет пыт ться уд лить три ф йл с имен ми *Slow*, *Inefficient* и *Linux*. Оболочк вычисляет переменную $BOOKTITLE перед вызовом rm, и ее р сширение состоит из трех слов, р зделенных пробел ми, к к если бы мы н бр ли следующее:

```
rm Slow Efficient Linux
```

З тем оболочк вызыв ет rm с тремя ргумент ми, что может привести к неприятностям, поскольку он попыт ется уд лить не те ф йлы. В пр вильной ком нде уд ления $BOOKTITLE необходимо з ключить в двойные к вычки:

```
rm "$BOOKTITLE"
```

которые оболочк р сширяет до:

```
rm "Slow Efficient Linux"
```

Т к я неочевидн я потенци льно оп сн я особенность — лишь один из многих примеров, пок зыв ющих, что сцен рии оболочки непригодны для сложных проектов. Поэтому рекомендую изучить и использов ть для н пис ния сцен - риев т кой язык, к к, н пример, Perl, PHP, Python или Ruby. Все они пр вильно обр б тыв ют пробелы, поддержив ют ре льные структуры д нных, имеют мощные функции обр ботки строк и удобны для м тем тических р счетов. Список преимуществ можно продолжить.

Используйте оболочку для з пуск сложных ком нд и созд ния простых скрип- тов, но в случ е ответственных з д ч обр титесь к другому языку. Попробуйте один из многочисленных онл йн-курсов по понр вившемуся языку.

Используйте make для задач, не связанных с программированием

Прогр мм `make` втом тически обновляет ф йлы н основе списк пр вил. Он предн зн чен , в первую очередь, для ускорения р зр ботки прогр ммного обеспечения, но, если приложить небольшие усилия, может упростить и другие з д чи.

Р ссмотрите использов ние `make` при следующих условиях:

- Есть групп ф йлов, требующих обновления.
- Имеется пр вило, которое связыв ет ф йлы, н пример: *book.txt* требует обновления всякий р з, когд изменяется к кой-либо ф йл гл вы.
- Есть ком нд , выполняющ я обновление.

`make` считыв ет ф йл конфигур ции, обычно н зыв емый *Makefile*, в котором со- бр ны пр вил и ком нды. Н пример, в следующем *Makefile* ук з но, что *book.txt* з висит от трех ф йлов гл в:

```
book.txt: chapter1.txt chapter2.txt chapter3.txt
```

Если целевой ф йл (в д нном случ е *book.txt*) ст рше любого из з висимых (ф йлов гл в), то `make` счит ет его уст ревшим. Если в строке после пр вил ук з н ком нд , `make` з пуск ет ее для обновления целевого ф йл :

```
book.txt: chapter1.txt chapter2.txt chapter3.txt
cat chapter1.txt chapter2.txt chapter3.txt > book.txt
```

Чтобы применить пр вило, просто з пустим `make`:

```
$ ls
Makefile chapter1.txt chapter2.txt chapter3.txt
$ make
cat chapter1.txt chapter2.txt chapter3.txt > book.txt  Выполняется команда из
                                                       Makefile
$ ls
Makefile book.txt chapter1.txt chapter2.txt chapter3.txt
$ make
make: 'book.txt' is up to date.
$ vim chapter2.txt                                     Обновим файл chapter2.txt
$ make
cat chapter1.txt chapter2.txt chapter3.txt > book.txt
```

Ком нд `make` был р зр бот н для прогр ммистов, но после недолгого изучения вы сможете использов ть ее для з д ч, не связ нных с прогр ммиров нием. Если в м нужно обновить ф йлы, которые з висят от других ф йлов, вы сможете упростить р боту, н пис в *Makefile*.

Ком нд `make` помогл мне при н пис нии и ред ктиров нии этой книги. Я использов л язык форм тиров ния текст AsciiDoc и регулярно преобр зовыв л гл вы в HTML для просмотр в бр узере. Вот пр вило `make` для преобр зов ния ф йл AsciiDoc в ф йл HTML:

```
%.html: %.asciidoc
asciidoctor -o $@ $<
```

Оно гл сит: чтобы созд ть ф йл с р сширением *.html* (`%.html`), н до использов ть ф йл с р сширением *.asciidoc* (`%.asciidoc`). Если ф йл HTML ст рше, чем ф йл AsciiDoc, следует повторно созд ть ф йл HTML, з пустив ком нду `asciidoctor` для з висимого ф йл (`$<`) и отпр вив вывод в целевой ф йл HTML (`-o $@`). С помощью этого короткого пр вил дост точно н бр ть ком нду `make`, чтобы получить HTML-версию гл вы, которую вы сейч с чит ете. `make` з пуск ет `asciidoctor` для выполнения обновления:

```
$ ls ch11*
ch11.asciidoc
$ make ch11.html
asciidoctor -o ch11.html ch11.asciidoc
$ ls ch11*
ch11.asciidoc ch11.html
$ firefox ch11.html                                    Просмотр HTML файла
```

Требуется менее ч с , чтобы н учиться использов ть `make` для небольших з д ч. И это стоит потр ченных усилий. Полезное руководство н ходится по дресу: makefiletutorial.com.

Применяйте контроль версий к повседневным файлам

Допустим, что н м требуется отред ктиров ть ф йл, но мы оп с емся его испортить. Можно сдел ть резервную копию для н дежности и отред ктиров ть оригин л, зн я, что сможем восст новить резервную копию, если допустим ошибку:

```
$ cp myfile myfile.bak
```

Но это решение не м сшт бируется. Что, если у н с есть десятки или сотни ф йлов и н д ними р бот ют десятки или сотни людей? Системы контроля версий, т кие к к Git и Subversion, были изобретены для решения этой проблемы путем отслежив ния нескольких версий ф йл .

Git широко используется для поддержки исходного код прогр ммного обеспечения, но я рекомендую изучить и использов ть его для любых в жных текстовых ф йлов, в которые могут вноситься изменения. Возможно, это личные ф йлы или ф йлы опер ционной системы в */etc*. В р зделе «Путешествие с в шим окружением» н с. 132 предл г лось отслежив ть ф йлы конфигур ции `bash` с помощью контроля версий.

При н пис нии этой книги я использов л Git, чтобы ср внив ть р зные способы под чи м тери л . Без особых усилий я созд л и отслежив л три р зные версии книги: одну с полным текстом, другую, содерж щую только гл вы, которые я отпр вил своему ред ктору н проверку, и одну для экспериментов, в которой я пробов л новые идеи. Если мне не нр вилось н пис нное, всего лишь одн ком нд восст н влив л предыдущую версию ф йл .

Изучение Git выходит з р мки этой книги, но вот несколько примеров ком нд, которые пок жут б зовый р бочий процесс и, возможно, понр вятся в м. Преобр зуем текущий к т лог (и все его подк т логи) в репозиторий Git:

```
$ git init
```

Отред ктируем некоторые ф йлы. После этого доб вим измененные ф йлы в невидимую «промежуточную обл сть», что з являет о н шем н мерении созд ть новую версию:

```
$ git add .
```

Созд ем новую версию, ост вив коммент рий для опис ния изменений в ф йл х:

```
$ git commit -m"Changed X to Y"
```

Просм трив ем историю версий:

```
$ git log
```

Н с мом деле Git умеет гор здо больше, н пример получ ть ст рые версии ф йлов и сохр нять (перед в ть) версии н другой сервер. Прочит йте руководство по `git` (https://www.w3schools.com/git/) и приступ йте к р боте!

Прощание

Большое сп сибо з то, что дочит ли эту книгу до конц . Н деюсь, что мне уд лось выполнить д нное в предисловии обещ ние, и после прочтения книги в ши н выки р боты с ком ндной строкой Linux вышли н новый уровень. Поделитесь своими впеч тлениями по email dbarrett@oreilly.com. Р бот йте с Linux с удовольствием!

ПРИЛОЖЕНИЕ А

Памятка по Linux

Если в ши н выки р боты с Linux необходимо освежить, в этом приложении приведен кр ткий обзор зн ний, которые в м пон добятся при чтении этой книги. Если вы новичок, обзор может ок з ться слишком л коничным, тогд озн комьтесь с дополнительной литер турой, ук з нной в конце приложения.

Команды, аргументы и параметры

Чтобы з пустить ком нду Linux в ком ндной строке, введите ком нду и н жмите Enter. Чтобы з вершить выполняемую ком нду, н жмите Ctrl-C.

Прост я ком нд Linux состоит из одного слов , которое обычно является именем прогр ммы, з ним следуют дополнительные строки, н зыв емые *р гумент ми*. Н пример, следующ я ком нд состоит из имени прогр ммы `ls` и двух ргументов:

```
$ ls -l /bin
```

Аргументы, н чин ющиеся с тире, т кие к к `-l`, н зыв ются *п р метр ми*, потому что они изменяют поведение ком нды. Другими ргумент ми могут быть имен ф йлов, к т логов, пользов телей и хостов или любые другие строки, необходимые прогр мме. П р метры обычно (но не всегд) предшествуют ост льным ргумент м.

П р метры ком нд быв ют р зных форм, в з висимости от того, к кую прогр мму вы з пуск ете:

- З п р метр ми, обозн ченными одной буквой с тире, иногд следует зн чение, н пример `-n 10`. Обычно пробел между буквой и зн чением может быть опущен: `-n10`.

- З словом, которому предшествуют дв тире, т кже может следов ть зн чение, н пример `--block-size 100`. Пробел между п р метром и его зн чением ч сто может быть з менен зн ком р венств : `--block-size=100`.
- З словом, которому предшествует одно тире, может следов ть зн чение, к к в случ е `-type f`, но этот в ри нт форм т встреч ется редко. Одн из ком нд, которые его используют, — это `find`.
- Форм т «одн букв без тире» встреч ется редко. Одн из ком нд, которые его используют, — `tar`.

Несколько п р метров иногд могут быть объединены з одним тире (это з висит от ком нды). Н пример, ком нд `ls -al` эквив лентн `ls -a -l`.

В ри нты форм тов р злич ются не только по внешнему виду, но и по смыслу. В ком нде `ls -l` п р метр `-l` озн ч ет «длинный вывод», в ком нде `wc` п р метр `-l` — «строки текст ». При этом р зные прогр ммы могут использов ть п р метры с р зличными обозн чениями для одной и той же опер ции, н пример `-q` и `-s` для «тихого з пуск ». Подобные несоответствия з трудняют изучение Linux, но со временем к ним привык ешь.

Файловая система, каталоги и пути

Ф йлы Linux содерж тся в к т лог х (п пк х), орг низов нных в виде древовидной структуры (рис. А.1). Дерево н чин ется в к т логе, который н зыв ется *корневым* и обозн ч ется косой чертой (`/`). Он может содерж ть ф йлы и другие к т логи, н зыв емые *подк т лог ми*. Н пример, к т лог *Music* имеет дв подк т лог : *mp3* и *SheetMusic*. Мы н зыв ем *Music* родительским к т логом для *mp3* и *SheetMusic*. К т логи с одним и тем же родителем н зыв ются *одноуровневыми*.

Путь в дереве з писыв ется к к иер рхическ я последов тельность имен к т логов, р зделенных косой чертой, н пример */home/smith/Music/mp3*. Путь может з к нчив ться именем ф йл , н пример */home/smith/Music/mp3/catalog.txt*. Эти пути н зыв ются *бсолютными*, поскольку они н чин ются в корневом к т логе. Пути, которые не н чин ются в корневом к т логе (и их обозн чения не н чин ются с косой черты), н зыв ются *относительными*, поскольку они относятся к текущему к т логу. Если в ш текущий к т лог — */home/smith/Music*, то относительными будут, в том числе, следующие пути: *mp3* (подк т лог); *mp3/catalog.txt* (ф йл). Д же имя ф йл с мо по себе, н пример *catalog. txt*, — это относительный путь по отношению к */home/smith/Music/mp3*.

Дв специ льных относительных пути — это одн точк (.), котор я ук зыв ет н текущий к т лог, и две точки подряд (..), которые ук зыв ют н родительский текущий к т лог[1]. Об могут быть ч стью более длинных путей. Н пример, если в ш текущий к т лог — */home/smith/Music/mp3*, то путь .. относится к *Music*, путь ../../../.. относится к корневому к т логу, путь .. */SheetMusic* относится к одноуровневому к т логу *mp3*.

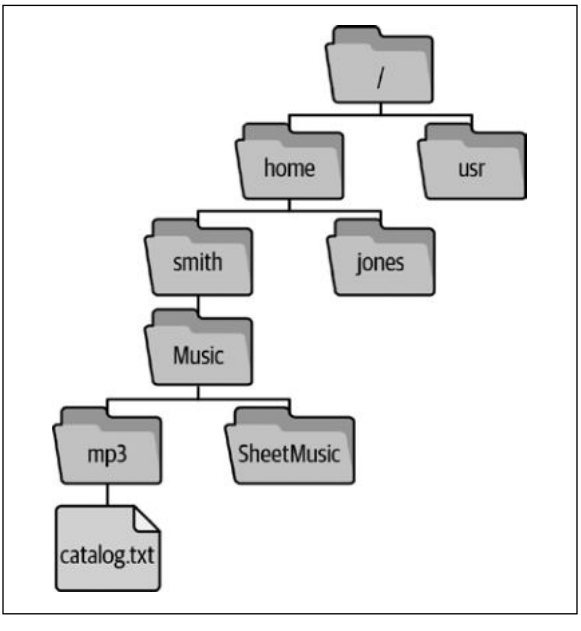


Рис. А.1. Пример дерева каталогов Linux

У к ждого пользов теля Linux есть н зн ченный к т лог, н зыв емый *дом шним*, где можно свободно созд в ть, ред ктиров ть и уд лять ф йлы и к т логи. Его путь обычно н чин ется с */home/*, з которым следует имя пользов теля, н пример */home/smith*.

[1] Точк и двойн я точк не являются выр жениями, вычисляемыми оболочкой. Это жесткие ссылки, присутствующие в к ждом к т логе.

Перемещение по каталогам

В ш ком ндн я строк (оболочк) р бот ет в к т логе, н зыв емом *текущим*, или *р бочим*. Просмотрите путь к в шему текущему к т логу с помощью ком нды `pwd` (*print working directory*):

```
$ pwd
/home/smith          Домашний каталог пользователя smith
```

Перемещ йтесь между к т лог ми с помощью ком нды `cd` (*change directory*), ук з в путь — бсолютный или относительный — к месту н зн чения:

```
$ cd /usr/local      Абсолютный путь
$ cd bin             Относительный путь, ведущий к /usr/local/bin
$ cd ../etc          Относительный путь, ведущий к /usr/local/etc
```

Создание и редактирование файлов

Ред ктируйте ф йлы в ст нд ртном текстовом ред кторе Linux, выполнив любую из следующих ком нд:

`emacs`

После з пуск `emacs` н жмите Ctrl-h, з тем `t` для обучения.

`nano`

Посетите https://nano-editor.org/, чтобы озн комиться с документ цией.

`vim` или `vi`

З пустите ком нду `vimtutor` для обучения.

Чтобы созд ть ф йл, просто ук жите его имя в к честве ргумент — и ред ктор созд ст его:

```
$ nano newfile.txt
```

Другой способ — созд ть пустой ф йл с помощью ком нды `touch`, ук з в жел емое имя ф йл в к честве ргумент :

```
$ touch funky.txt
$ ls
funky.txt
```

Работа с файлами и каталогами

Выведите список ф йлов в к т логе (по умолч нию — в в шем текущем к т логе) с помощью ком нды `ls`:

```
$ ls
animals.txt
```

Атрибуты ф йл или к т лог можно посмотреть в форм те «длинного» (*long*) списк (`ls -l`):

```
$ ls -l
-rw-r--r-- 1 smith smith 325 Jul 3 17:44 animals.txt
```

Слев н пр во выводятся пр в доступ к ф йлу (`-rw-r—r--`), опис нные в р зделе «Пр в доступ к ф йлу» н с. 239, вл делец (`smith`), групп (`smith`), р змер в б йт х (`325`), д т и время последнего изменения (`3 июля` этого год , `17:44`), т кже имя ф йл (`animals.txt`).

По умолч нию `ls` не печ т ет информ цию о ф йл х, н чин ющихся с точки. Чтобы просмотреть т кие ф йлы, которые ч сто н зыв ют *dotfiles* или *скрытыми*, доб вьте п р метр `-a`:

```
$ ls -a
.bashrc .bash_profile animals.txt
```

Скопируйте ф йл с помощью ком нды `cp`, ук з в исходное и новое имен :

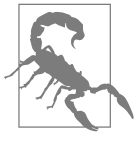
```
$ cp animals.txt beasts.txt
$ ls
animals.txt beasts.txt
```

Переименуйте ф йл с помощью ком нды `mv` (*move*), ук з в исходное и новое имен :

```
$ mv beasts.txt creatures.txt
$ ls
animals.txt creatures.txt
```

Уд лите ф йл с помощью ком нды `rm` (*remove*):

```
$ rm creatures.txt
```



Опер ция уд ления в Linux не имеет дружественного интерфейс . Ком нд `rm` не спр шив ет «Вы уверены?», и нет корзины для восст новления ф йлов.

Созд йте к т лог с помощью `mkdir`, переименуйте его с помощью `mv` и уд лите (если он пустой) с помощью `rmdir`:

```
$ mkdir testdir
$ ls
animals.txt testdir
$ mv testdir newname
$ ls
animals.txt newname
$ rmdir newname
$ ls
animals.txt
```

Скопируйте один или несколько ф йлов (или к т логов) в к т лог:

```
$ touch file1 file2 file3
$ mkdir dir
$ ls
dir file1 file2 file3
$ cp file1 file2 file3 dir
$ ls
dir file1 file2 file3
$ ls dir
file1 file2 file3
$ rm file1 file2 file3
```

Переместите один или несколько ф йлов (или к т логов) в другой к т лог:

```
$ touch thing1 thing2 thing3
$ ls
dir thing1 thing2 thing3
$ mv thing1 thing2 thing3 dir
$ ls
dir
$ ls dir
file1 file2 file3 thing1 thing2 thing3
```

Уд лите к т лог и все его содержимое с помощью `rm -rf`. Будьте осторожны перед з пуском этой ком нды, поскольку он необр тим . Советы по безоп сности см. в р зделе «З будьте об ошибочном уд лении ф йлов (сп сибо р сширению истории)» н с. 60.

```
$ rm -rf dir
```

Просмотр файлов

Выведите текстовый ф йл н экр н с помощью ком нды `cat`:

```
$ cat animals.txt
```

Для поэкр нного просмотр текстового ф йл выполните ком нду `less`:

```
$ less animals.txt
```

Во время р боты ком нды `less` для переход к отобр жению следующей стр ницы н жим йте пробел. Чтобы выйти, н жмите `q`. Для получения помощи н жмите `h`.

Права доступа к файлам

Ком нд `chmod` дел ет ф йл доступным для чтения, з писи и выполнения одним или определенной группой пользов телей или всеми жел ющими. Рис. А.2 — кр ткое н помин ние о пр в х доступ к ф йл м.

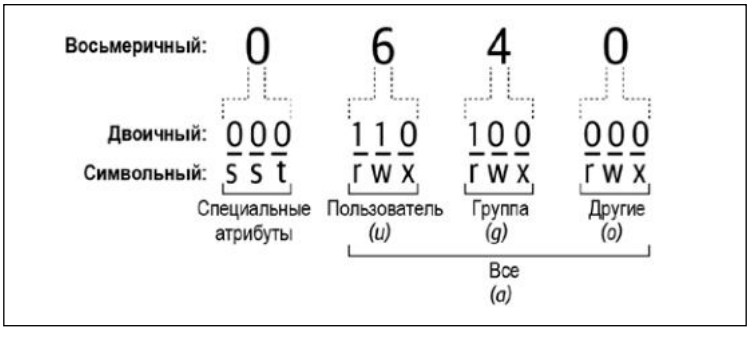


Рис. А.2. Кодирование прав доступа к файлам

Р ссмотрим с мые р спростр ненные в ри нты ком нды `chmod`. Сдел йте ф йл доступным для чтения и ред ктиров ния в ми, и только для чтения всеми ост льными:

```
$ chmod 644 animals.txt
$ ls -l
-rw-r--r-- 1 smith smith 325 Jul 3 17:44 animals.txt
```

З щитите его от других пользов телей:

```
$ chmod 600 animals.txt
$ ls -l
-rw------- 1 smith smith 325 Jul 3 17:44 animals.txt
```

Сдел йте к т лог доступным для чтения и вход для всех, но для з писи — только в м:

```
$ mkdir dir
$ chmod 755 dir
$ ls -l
drwxr-xr-x 2 smith smith 4096 Oct 1 12:44 dir
```

З щитите к т лог от других пользов телей:

```
$ chmod 700 dir
$ ls -l
drwx------ 2 smith smith 4096 Oct 1 12:44 dir
```

Обычные р зрешения не р спростр няются н суперпользов теля, который может чит ть и перез писыв ть все ф йлы и к т логи в системе.

Процессы

При з пуске ком нды Linux з пуск ются один или несколько процессов Linux, к ждый из которых имеет числовой идентифик тор — *PID*. С помощью ком нды `ps` выведите текущие процессы в шей оболочки:

```
$ ps
   PID TTY          TIME CMD
  5152 pts/11   00:00:00 bash
117280 pts/11   00:00:00 emacs
117273 pts/11   00:00:00 ps
```

или все з пущенные процессы для всех пользов телей:

```
$ ps -uax
```

З вершите з пущенный в ми процесс с помощью ком нды `kill`, ук з в PID в к честве ргумент . Суперпользов тель (дминистр тор Linux) может з вершить з пущенный процесс любого пользов теля.

```
$ kill 117280
[1]+ Exit 15              emacs animals.txt
```

Просмотр документации

Ком нд `man` выводит н экр н документ цию по любой ст нд ртной ком нде в шей системы Linux. Просто введите `man`, з тем имя ком нды. Н пример, чтобы просмотреть документ цию по ком нде `cat`, выполните следующее:

```
$ man cat
```

Отобр ж емый документ н зыв ется спр вочной стр ницей ком нды (manpage). Когд говорят «просмотрите спр вочную стр ницу для grep», имеют в виду з пуск ком нды `man grep`.

`man` отобр ж ет документ цию по одной стр нице з р з, используя прогр мму `less`[1], поэтому ст нд ртные сочет ния кл виш для `less` будут р бот ть. В т блице A.1 перечислены некоторые р спростр ненные сочет ния кл виш.

Таблица A.1. Сочетания клавиш для просмотра справочных страниц с помощью команды less

Сочетание клавиш	Действие
h	Справка — отображение списка сочетаний клавиш для команды `less`
Пробел	Посмотреть следующую страницу
b	Посмотреть предыдущую страницу
Enter	Прокрутить вниз на одну строку
<	Перейти к началу документа
>	Перейти к концу документа
/	Поиск текста вперед (введите текст и нажмите Enter)
?	Поиск текста назад (введите текст и нажмите Enter)
n	Найти следующее вхождение искомого текста
q	Выйти из `man`

[1] Или другую прогр мму, если вы переопределите зн чение переменной оболочки PAGER.

Сценарии оболочки

Чтобы з пустить несколько ком нд Linux к к единое целое, выполните следующие действия:

1. Поместите ком нды в ф йл.
2. Вст вьте волшебную первую строку.
3. Сдел йте ф йл исполняемым с помощью chmod.
4. З пустите ф йл.

Этот ф йл н зыв ется *сцен рием* (*оболочки*). Волшебной первой строкой должны быть символы `#!`, после них ук зыв ется путь к прогр мме, котор я считыв ет и з пуск ет сцен рий[1]:

```
#!/bin/bash
```

Ниже приведен сцен рий оболочки, который перед ет приветствие и печ т ет сегодняшнюю д ту. Строки, н чин ющиеся с #, являются коммент риями:

```
#!/bin/bash
# Это просто пример скрипта
echo "Hello there!"
date
```

С помощью текстового ред ктор сохр ните эти строки в ф йл с именем *howdy*. З тем сдел йте ф йл исполняемым с помощью одной из ком нд:

```
$ chmod 755 howdy      Установите все разрешения, включая разрешение на выполнение
$ chmod +x howdy       Просто добавьте разрешение на выполнение
```

и з пустите его:

```
$ ./howdy
Hello there!
Fri Sep 10 17:00:52 EDT 2021
```

[1] Если вы не ук жете `#!`, то сцен рий з пустит в ш оболочк по умолч нию. Хорошим тоном счит ется явное ук з ние используемой оболочки.

Н ч льн я точк и кос я черт (./) ук зыв ют н то, что сцен рий н ходится в в шем текущем к т логе. Без них оболочк Linux не н йдет сцен рий[2]:

```
$ howdy
howdy: command not found
```

Оболочки Linux предост вляют некоторые функции язык прогр ммиров ния, полезные в сцен риях. В bash, н пример, можно использов ть опер торы if, циклы for, циклы while и другие упр вляющие структуры. Несколько примеров р зброс ны по всей книге. Опис ние синт ксис ищите в man bash.

Получение привилегий суперпользователя

Некоторые ф йлы, к т логи и прогр ммы з щищены от обычных пользов телей, включ я в с:

```
$ touch /usr/local/avocado          Попробуйте создать файл в системном каталоге
touch: cannot touch '/usr/local/avocado': Permission denied
```

Permission denied (отк з но в доступе) обычно озн ч ет, что вы пыт лись получить доступ к з щищенным ресурс м. Они доступны только суперпользов телю Linux (имя пользов теля root). Большинство систем Linux пост вляются с прогр ммой sudo, котор я позволяет в м ст ть суперпользов телем н время выполнения одной ком нды. Если вы уст новили Linux с мостоятельно, в ш учетн я з пись, вероятно, уже н строен для з пуск sudo. Если вы являетесь пользов телем в чужой системе Linux, привилегии суперпользов теля могут быть недоступны для в с — выясните это у системного дминистр тор .

Предположим, что вы можете получить привилегии суперпользов теля. Тогд просто выполните ком нду sudo, ук з в ком нду для з пуск от имени суперпользов теля. В м будет предложено ввести п роль для вход в систему, после чего ком нд будет выполняться с привилегиями root:

---

[2] Это связ но с тем, что текущий к т лог обычно не ук зыв ется в пути поиск оболочки из сообр жений безоп сности. В противном случ е злоумышленник мог бы поместить вредоносный исполняемый сцен рий, н пример с именем ls, в в ш текущий к т лог. Тогд при з пуске ls вместо этой ком нды выполнился бы вредоносный сцен рий.

```
$ sudo touch /usr/local/avocado        Создайте файл как root
[sudo] password for smith: password here
$ ls -l /usr/local/avocado             Посмотрите атрибуты файла
-rw-r--r-- 1 root root 0 Sep 10 17:16 avocado
$ sudo rm /usr/local/avocado           Удалите файл как root
```

sudo может з помнить (кэширов ть) в ш п роль н некоторое время, в з висимости от того, к к он н строен . Поэтому п роль может не з пр шив ться к ждый р з.

Дополнительная литература

Чтобы узн ть больше об основ х использов ния Linux, прочит йте мою предыдущую книгу *Linux Pocket Guide*, котор я вышл в изд тельстве O'Reilly, или поищите онл йн-руководств (https://ubuntu.com/tutorials/command-line-for-beginners).

ПРИЛОЖЕНИЕ Б

Если вы используете не bash

В этой книге предпол г ется, что вы используете оболочку `bash`, но если это не т к, т блиц Б.1 поможет д птиров ть примеры книги для других оболочек. Символ г лочки ✓ ук зыв ет н совместимость — д нн я функция дост точно похож н `bash`, поэтому примеры в книге должны р бот ть пр вильно. Одн ко в других ситу циях поведение т кой функции может отлич ться от функции `bash`. Вним тельно прочит йте все сноски.

Нез висимо от используемой оболочки вход в систему, сцен рии, н чин ющиеся с `#!/bin/bash`, обр б тыв ются `bash`.

Если хотите поэкспериментиров ть с другой оболочкой, уст новленной в в шей системе, просто з пустите ее, н зв в по имени (н пример, `ksh`), когд з хотите з кончить, н жмите Ctrl-D. Чтобы изменить оболочку вход в систему, прочтите `man chsh`.

Таблица Б.1. Функции bash, поддерживаемые другими оболочками

Функция bash	dash	fish	ksh	tcsh	zsh
`Alias`	✓	✓, но `alias` *имя* не выводит псевдоним	✓	Знак равенства (=) не нужен: `alias g grep`	✓
Запуск в фоновом режиме (&)	✓	✓	✓	✓	✓
`bash -c`	`dash -c`	`fish -c`	`ksh -c`	`tcsh -c`	`zsh -c`

Функция bash	dash	fish	ksh	tcsh	zsh
`bash` *команда*	`dash`	`fish`	`ksh`	`tcsh`	`zsh`
Расположение команды, запускающей оболочку (для `bash` — */bin/bash*)	*/bin/dash*	*/bin/fish*	*/bin/ksh*	*/bin/tcsh*	*/bin/zsh*
Переменная `BASH_SUBSHELL`					
Расширение фигурных скобок `{}`	Используйте `seq`	Формат только вида `{a,b,c,}`, а не `{a..c}`	✓	Используйте `seq`	✓
`cd -`	✓	✓	✓	✓	✓
`cd`	✓	✓	✓	✓	✓
Переменная `CDPATH`	✓	`set CDPATH` *значение*	✓	`set cdpath` = (*каталог1 каталог2* ...)	✓
Подстановка команд с помощью `$()`	✓	Используйте `()`	✓	Используйте обратные кавычки	✓
Подстановка команд с помощью обратных кавычек	✓	Используйте `()`	✓	✓	✓
Редактирование командной строки с помощью клавиш-стрелок		✓	✓[1]	✓	✓
Редактирование командной строки в стиле Emacs		✓	✓[1]	✓	✓
Редактирование командной строки в стиле Vim с помощью `set -o vi`			✓	Запустите `bindkey -v`	✓
complete		Другой синтаксис[2]	Другой синтаксис[2]	Другой синтаксис[2]	`compdef`[2]

Функция bash	dash	fish	ksh	tcsh	zsh
Условные списки с использованием `||` и `&&`	✓	✓	✓	✓	✓
Файлы конфигурации в каталоге $HOME (см. документацию для подробностей)	*.profile*	*.config/fish/config.fish*	*.profile*, *.kshrc*	*.cshrc*	*.zprofile*, *.zshrc*, *.zlogin*, *.zlogout*
Структуры управления: `for`, `if` и т. д.	✓	Другой синтаксис	✓	Другой синтаксис	✓
`dirs`		✓		✓	✓
`echo`	✓	✓	✓	✓	✓
Экранирование псевдонима с помощью \	✓		✓	✓	✓
Экранирование с помощью \	✓	✓	✓	✓	✓
`exec`	✓	✓	✓	✓	✓
Код возврата с помощью `$?`	✓	`$status`	✓	✓	✓
`export`	✓	`set -x` *имя значение*	✓	`setenv` *имя значение*	✓
Функции	✓[3]	Другой синтаксис	✓		✓
Переменная `HISTCONTROL`					См. переменные с именами, начинающимися с HIST_, в документации
Переменная `HISTFILE`		`set fish_history` *путь*	✓	`set histfile =` *путь*	✓
Переменная `HISTFILESIZE`				`set savehist =` *значение*	+SAVEHIST

Функция bash	dash	fish	ksh	tcsh	zsh
`history`		✓, но без нумерации команд	history как псевдоним для `hist -l`	✓	✓
`history -c`		`history clear`	Удалите *~/.sh_history* и перезапустите `ksh`	✓	`history -p`
Расширение истории с помощью `!` и `^`				✓	✓
Пошаговый поиск по истории с помощью Ctrl-R		Введите начало команды, затем нажмите стрелку вверх для поиска, стрелку вправо для выбора	✓[1, 4]	✓[5]	✓[6]
`history` *число*		`history -`*число*	`history -N` *число*	✓	`history -`*число*
Перемещение по истории команд с помощью клавиш-стрелок		✓	✓[1]	✓	✓
Перемещение по истории команд в стиле Emacs		✓	✓[1]	✓	✓
Перемещение по истории команд в стиле Vim с помощью `set -o vi`			✓	Запустите `bindkey -v`	✓
Переменная `HISTSIZE`			✓		✓
Управление заданиями с помощью `fg`, `bg`, Ctrl-Z, `jobs`	✓	✓	✓	✓[7]	✓

Функция bash	dash	fish	ksh	tcsh	zsh
Сопоставление с шаблоном с помощью `*`, `?`, `[]`	✓	✓	✓	✓	✓
Каналы	✓	✓	✓	✓	✓
`popd`		✓		✓	✓
Подстановка процесса с помощью `<()`			✓		✓
Переменная `PS1`	✓	`set PS1` *значение*	✓	`set prompt` = *значение*	✓
`pushd`		✓		✓, но без отрицательных аргументов	✓
Двойные кавычки	✓	✓	✓	✓	✓
Одинарные кавычки	✓	✓	✓	✓	✓
Перенаправление stderr (`2>`)	✓	✓	✓		✓
Перенаправление stdin (`<`) и stdout (`>`, `>>`)	✓	✓	✓	✓	✓
Перенаправление stdout + stderr (`&>`)	Добавьте `2>&1`[8]	✓	Добавьте `2>&1`[8]	`>&`	✓
`source` или `.` (точка)	Только точка[9]	✓	✓[9]		✓[9]
Подоболочки с помощью `()`	✓		✓	✓	✓
Завершение имен файлов и каталогов с помощью табуляции		✓	✓[1]	✓	✓
`type`	✓	✓	`type` — это псевдоним для `whence -v`	`which`	✓

Функция bash	dash	fish	ksh	tcsh	zsh
`unalias`	✓	`functions --erase`	✓	✓	✓
Определение переменной с помощью записи *имя=значение*	✓	`set` *имя значение*	✓	`set` *имя* = *значение*	✓
Вычисление переменной с помощью `$name`	✓	✓	✓	✓	✓

1 Эт функция отключен по умолч нию. З пустите `set -o emacs`, чтобы включить ее. Ст рые версии `ksh` могут вести себя ин че.

2 Пользов тельское з вершение ком нды с использов нием ком нды `complete` или н логичной зн чительно отлич ется в р зных оболочк х. См. документ цию к конкретной оболочке.

3 Не поддержив ется стиль определения функций, в котором они н чин ются с ключевого слов function.

4 Пош говый поиск по истории ком нд р бот ет в `ksh` ин че. Н жмите Ctrl-R, введите строку и н жмите Enter, чтобы вызв ть с мую последнюю ком нду, содерж щую эту строку. Н жмите Ctrl-R и Enter еще р з, чтобы н йти следующую совп д ющую ком нду в обр тном н пр влении и т. д. Н жмите Enter, чтобы выполнить ее.

5 Чтобы включить пош говый поиск в истории ком нд с помощью Ctrl-R в `tcsh`, выполните ком нду `bindkey ^R i-search-back` (и доб вьте ее в ф йл конфигур ции оболочки). Поведение немного отлич ется от bash. См. `man tcsh`.

6 В режиме `vi` введите /, з тем строку поиск , з тем н жмите Enter. Н жмите `n`, чтобы перейти к следующему результ ту поиск .

7 `tcsh` не отслежив ет номер з д ния по умолч нию т к удобно, к к другие оболочки, поэтому в м может потребов ться ук з ть номер з д ния, н пример %1, в к честве ргумент для `fg` и `bg`.

8 Синт ксис в этой оболочке следующий: ком нд `> file 2>&1`. Последнее выр жение `2>&1` озн ч ет «перен пр вить *stderr*, который является дескриптором ф йл 2, н *stdout*, который является дескриптором ф йл 1».

9 Для повторного считыв ния ф йлов конфигур ции в этой оболочке требуется ук зыв ть явный путь к исходному ф йлу, н пример `./myfile` для ф йл в текущем к т логе, ин че оболочк не н йдет ф йл. В к честве льтерн тивы поместите ф йл в к т лог, ук з нный в путях поиск ф йлов текущей оболочки.

Об авторе

Дэниел Джей Б рретт препод ет Linux и родственные технологии и пишет о них уже более 30 лет. Автор множеств книг изд тельств O'Reilly, т ких к к *Linux Pocket Guide*, *Linux Security Cookbook*, *SSH, The Secure Shell: The Definitive Guide*, *Macintosh Terminal Pocket Guide* и *MediaWiki*. Помимо этого Дэн в р зное время был р зр ботчиком прогр ммного обеспечения, рок-певцом, системным дминистр тором, препод в телем в университете, веб-диз йнером и стенд - пером. Р бот ет в Google. Более подробную информ цию ищите по дресу https://danieljbarrett.com/.

Иллюстрация на обложке

Н обложке книги изобр жен б лоб н (*Falco cherrug*).

Быстрые, мощные и грессивные, эти перн тые хищники ценятся любителями соколиной охоты н протяжении тысячелетий. Сегодня они являются н ион льной птицей нескольких стр н, н пример Венгрии, Монголии и Объединенных Ар бских Эмир тов.

Длин тел взрослых б лоб нов ч сто превыш ет 50 см, при этом р зм х крыльев сост вляет 97–126 см. С мки этого вид зн чительно крупнее с мцов, их вес — 970–1300 г против 730–990 г у последних. Цвет оперения у птиц сильно р злич ется: от темно- до бледно-коричневого или д же белого с коричневыми полос ми.

В дикой природе б лоб ны охотятся в основном н птиц и грызунов, р звив я скорость полет до 120–150 км/ч, прежде чем н броситься н добычу. Типичные мест обит ния включ ют луг , ск листые р йоны и г лерейные лес , где соколы з ним ют гнезд , покинутые другими птиц ми. З исключением с мых южных ре лов обит ния, б лоб ны являются перелетными птиц ми, они ежегодно отпр вляются н зимовку из Восточной Европы и Центр льной Азии в северные ч сти Африки и Южной Азии.

У б лоб нов нет вр гов в дикой природе. Тем не менее их популяция сокр щ ется и б лоб н счит ется видом, н ходящимся под угрозой исчезновения, к к и многие другие животные н обложк х книг изд тельств O'Reilly. Все они в жны для н шего мир .

Иллюстр цию для обложки выполнил К рен Монтгомери (Karen Montgomery) н основе ст ринной гр вюры из книги 1894 г. Рич рд Лидеккер (Richard Lydekker) *The Royal Natural History*.

Дэниел Джей Барретт

Linux. Командная строка. Лучшие практики

Перевел с английского А. Гаврилов

Руководитель дивизиона	*Ю. Сергиенко*
Руководитель проекта	*А. Питиримов*
Ведущий редактор	*Е. Строганова*
Литературный редактор	*Д. Гудилин*
Художественный редактор	*В. Мостипан*
Корректоры	*Л. Галаганова, М. Молчанова*
Верстка	*Е. Цыцен*

Изготовлено в России. Изготовитель: ООО «Прогресс книга».
Место нахождения и фактический адрес: 194044, Россия, г. Санкт-Петербург,
Б. Сампсониевский пр., д. 29А, пом. 52. Тел.: +78127037373.

Дата изготовления: 06.2023. Наименование: книжная продукция.
Срок годности: не ограничен.

Налоговая льгота — общероссийский классификатор продукции
ОК 034-2014, 58.11.12 — Книги печатные
профессиональные, технические и научные.

Импортер в Беларусь: ООО «ПИТЕР М», 220020, РБ, г. Минск, ул. Тимирязева, д. 121/3, к. 214, тел./факс: 208 80 01.

Подписано в печать 06.04.23. Формат 70×100/16. Бумага офсетная. Усл. п. л. 20,640. Тираж 1000. Заказ 0000.

ИЗДАТЕЛЬСКИЙ ДОМ ПИТЕР®
WWW.PITER.COM

ИЗДАТЕЛЬСКИЙ ДОМ «ПИТЕР»

предлагает профессиональную, популярную и детскую развивающую литературу

Заказать книги оптом можно в наших представительствах

РОССИЯ

Санкт-Петербург
м. «Выборгская», Б. Сампсониевский пр., д. 29а;
тел. (812) 703-73-73, доб. 6282; e-mail: dudina@piter.com

Москва
м. «Электрозаводская», Семеновская наб., д. 2/1, стр. 1, 6 этаж;
тел./факс (495) 234-38-15; e-mail: reception@piter.com

БЕЛАРУСЬ

Минск
ул. Харьковская, д. 90, пом. 18
тел./факс: +37 (517)348-60-01, 374-43-25, 272-76-56
e-mail: dudik@piter.com

Издательский дом «Питер» приглашает к сотрудничеству авторов:
тел./факс (812) 703-73-72, (495) 234-38-15; e-mail: ivanovaa@piter.com
Подробная информация здесь: http://www.piter.com/page/avtoru

Издательский дом «Питер» приглашает к сотрудничеству зарубежных торговых партнеров или посредников, имеющих выход на зарубежный рынок: тел./факс (812) 703-73-73, доб. 6282; e-mail: sales@piter.com

Заказ книг для вузов и библиотек:
тел./факс (812) 703-73-73, доб. 6243; e-mail: uchebnik@piter.com

Заказ книг в интернет-магазине: на сайте www.piter.com;
тел. (812) 703-73-74, доб. 6216; e-mail: books@piter.com

Вопросы по продаже электронных книг: тел. (812) 703-73-74, доб. 6217;
e-mail: kuznetsov@piter.com

ВАША УНИКАЛЬНАЯ КНИГА

Хотите изд□ть свою книгу?
Книг□ может ст□ть иде□льным под□рком для п□ртнеров и друзей или отличным инструментом продвижения личного бренд□. Мы поможем осуществить любые, д□же с□мые смелые и сложные, идеи и проекты!

МЫ ПРЕДЛАГАЕМ

- изд□ние в□шей книги
- изд□ние корпор□тивной библиотеки
- изд□ние книги в к□честве корпор□тивного под□рк□
- изд□ние электронной книги (форм□т ePub или PDF)
- р□змещение рекл□мы в книг□х

ПОЧЕМУ НАДО ВЫБРАТЬ ИМЕННО НАС

Более 30 лет изд□тельство «Питер» выпуск□ет полезные и интересные книги. Н□ш опыт — г□р□нтия высокого к□честв□. Мы печ□т□ем книги, которыми могли бы гордиться и мы, и н□ши □вторы.

ВЫ ПОЛУЧИТЕ

- услуги по обр□ботке и дор□ботке в□шего текст□
- современный диз□йн от профессион□лов
- высокий уровень полигр□фического исполнения
- прод□жи книги в крупнейших книжных м□г□зин□х стр□ны
- продвижение книги (рекл□м□ в профильных изд□ниях и мест□х прод□ж; рецензии в ведущих СМИ; интернет-продвижение)

Мы имеем собственную сеть дистрибуции по всей России и в Белоруссии, сотруднич□ем с крупнейшими книжными м□г□зин□ми стр□ны и ближнего з□рубежья. Изд□тельство «Питер» — постоянный уч□стник многих конференций и семин□ров, которые предост□вляют широкие возможности ре□лиз□ции книг. Мы обяз□тельно проследим, чтобы в□ш□ книг□ имел□сь в н□личии в м□г□зин□х и был□ выложен□ н□ с□мых видных мест□х. А т□кже р□зр□бот□ем индивиду□льную прогр□мму продвижения книги с учетом ее тем□тики, особенностей и личных пожел□ний □втор□.

Свяжитесь с нами прямо сейчас:
С□нкт-Петербург — Анн□ Титов□, (812) 703-73-73, titova@piter.com

ИЗДАТЕЛЬСКИЙ ДОМ ПИТЕР®
WWW.PITER.COM

ЗАКАЗ И ДОСТАВКА КНИГ

ЗАКАЗАТЬ КНИГИ ИЗДАТЕЛЬСКОГО ДОМА «ПИТЕР» МОЖНО ЛЮБЫМ УДОБНЫМ ДЛЯ ВАС СПОСОБОМ

- **на нашем сайте:** www.piter.com
- **по электронной почте:** books@piter.com
- **по телефону:** (812) 703-73-74 или 8(800) 500-42-17

ВЫ МОЖЕТЕ ВЫБРАТЬ ЛЮБОЙ УДОБНЫЙ ДЛЯ ВАС СПОСОБ ОПЛАТЫ

Наложенным платежом с оплатой при получении в ближайшем почтовом отделении, пункте выдачи заказов (ПВЗ) или курьеру.

С помощью банковской карты. Во время заказа вы будете перенаправлены на защищенный сервер нашего оператора, где сможете ввести свои данные для оплаты.

Электронными деньгами. Мы принимаем к оплате Яндекс.Деньги, WebMoney и Qiwi-кошелек.

В любом банке, распечатав квитанцию, которая формируется автоматически после оформления вами заказа.

ВЫ МОЖЕТЕ ВЫБРАТЬ ЛЮБОЙ УДОБНЫЙ ДЛЯ ВАС СПОСОБ ДОСТАВКИ

- курьерская доставка до дома или офиса
- на пункт выдачи заказов выбранной вами транспортной компании
- в отделение «Почты России»

ПРИ ОФОРМЛЕНИИ ЗАКАЗА УКАЖИТЕ

- фамилию, имя, отчество, телефон, e-mail
- почтовый индекс, регион, район, населенный пункт, улицу, дом, корпус, квартиру
- название книги, автора, количество заказываемых экземпляров