Команды обычно не знают, что они являются частью конвейера. `ls` считает, что выводит данные на дисплей, хотя на самом деле ее вывод был перенаправлен на `less`. А `less` верит, что читает данные с клавиатуры, когда на самом деле получает вывод `ls`.

Шесть команд для начала

Использование каналов — неотъемлемая часть работы с Linux. Давайте развивать ваши навыки работы с каналами с помощью небольшого набора команд Linux. Когда вы столкнетесь с какими-то из этих команд в будущем, то будете готовы их объединять.

Команды `wc, head, cut, grep, sort` и `uniq` имеют множество опций и режимов работы, про которые пока умолчим, чтобы сосредоточиться на каналах. Если хотите узнать больше о конкретной команде, запустите команду `man`, которая отобразит полную документацию. Например:

```
$ man wc
```

Для демонстрации этих шести команд в действии будем использовать файл с именем *animals.txt*, в котором перечислены некоторые сведения из книг издательства O'Reilly. Содержимое файла представлено 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
```

Каждая строка содержит четыре факта о какой-либо книге издательства O'Reilly, разделенные одним символом табуляции: животное на обложке, название книги, год публикации и имя первого автора.

Команда #1: wc

Команда `wc` выводит количество строк, слов и символов в файле:

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

wc сообщает, что в файле *animals.txt* 7 строк, 51 слово и 325 символов. Если вы посчитаете символы, включая пробелы и табуляции, то найдете только 318 символов, но wc также учитывает скрытые символы новой строки.

Опции -l, -w и -c указывают wc печатать 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 параметров текста — настолько полезная и часто встречающаяся задача, что авторы wc разработали свою команду специально для работы с каналами. Если вы не указываете имя файла, она читает данные из стандартного ввода и отправляет данные в стандартный вывод. Давайте воспользуемся ls для вывода содержимого текущего каталога и используем wc для подсчета строк. Этот конвейер отвечает на вопрос «Сколько файлов в моем текущем каталоге?»:

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

Опция -1, указывающая ls выводить результаты в один столбец, здесь не обязательна. Чтобы узнать, почему мы ее использовали, см. врезку «ls меняет свое поведение при перенаправлении вывода» на с. 23.

wc — это первая команда, с которой мы познакомились в этой главе, поэтому пока сложно использовать каналы. Ради интереса направим вывод wc самой себе, чтобы продемонстрировать, что одна и та же команда может появляться в конвейере более одного раза. Эта комбинированная команда сообщает, что количество слов в выводе wc равно четырем (три целых числа и имя файла):

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

Не будем останавливаться. Добавим третий вызов wc в конвейер и посчитаем 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 большинства других команд Linux, `ls` знает, выводит ли она данные на экран или перенаправляет в канал. Когда *stdout* — это экран, `ls` упорядочивает вывод в несколько колонок для удобства чтения:

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

При перенаправлении *stdout* `ls` создает одну колонку. Убедимся в этом, передав вывод `ls` команде, которая воспроизводит свой ввод, такой как `cat`[1]:

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

Это может привести к неожиданным результатам, как в следующем примере:

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

Первая команда `ls` выводит все имена файлов в одну строку, а вторая сообщает, что выводятся четыре строки. Если вы не знаете об особых свойствах `ls`, это несоответствие может удивить.

`ls` позволяет переопределить свое поведение по умолчанию. Вы можете заставить `ls` вывести на экран одну колонку с помощью параметра `-1` или несколько колонок с параметром `-C`.

Вывод указывает на одну строку (содержащую число 4), одно слово (само число 4) и два символа. Почему два? Потому что строка *4* заканчивается скрытым символом новой строки.

Думаю, что достаточно простейших конвейеров с `wc`. По мере того как мы изучим больше команд, конвейеры станут более практичными.

[1] В зависимости от ваших настроек `ls` может использовать также другие функции форматирования, например изменение цвета при выводе на экране. Но не при перенаправлении вывода.

Команда #2: head

Команда `head` выводит первые строки файла. Выведем первые три строки файла *animals.txt*, используя `head` с параметром `-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 запрашиваете больше строк, чем содержится в файле, `head` выведет весь файл (как это делает `cat`). Если вы опустите параметр `-n`, заголовок по умолчанию будет состоять из 10 строк (`-n10`).

Сама по себе функция `head` удобна для просмотра начала файла, когда вас не интересует остальное содержимое. Эта команда работает быстро и эффективно даже с очень большими файлами, поскольку ей не нужно считывать весь файл. Кроме того, `head` умеет считывать данные из стандартного ввода и использует стандартный вывод, что делает ее полезной в конвейерах. Подсчитаем количество слов в первых трех строках файла *animals.txt*:

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

Обычное использование `head` заключается в сокращении вывода данных от другой команды, когда вам не нужно видеть полную информацию, например длинный список каталогов. Перечислим первые пять имен файлов в каталоге */bin*:

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

Команда #3: cut

Команда `cut` выводит одну или несколько колонок из файла. Например, выведем все названия книг, которые расположены во второй колонке файла *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
```

Команда `cut` поддерживает два способа определения, что считать колонкой. Первый — разделение по полям (`-f`), когда входные данные состоят из строк (полей), каждая из которых разделена одним символом табуляции. Именно такой формат используется в файле *animals.txt*. Команда `cut` из предыдущего примера печатает второе поле каждой строки благодаря опции `-f2`.

Чтобы сократить вывод, передадим его в `head`. Выведем на экран только первые три строки:

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

Вы также можете вырезать несколько полей, разделив их номера запятыми:

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

или указав диапазон значений:

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

Также можно определить колонку для команды `cut` по положению символа в строке с использованием параметра `-c`. Выведем первые три символа из каждой строки файла, которые можно указать либо через запятую (1, 2, 3), либо в формате диапазона (1–3):

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

Теперь, когда вы ознакомились с основными функциями, попробуем сделать что-нибудь более практичное с помощью команды `cut` и с использованием каналов. Допустим, что файл *animals.txt* состоит из нескольких тысяч строк

и вам нужно извлечь только фамилии авторов. Сначала выделим четвертое поле — имя и фамилия автора:

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

Затем передадим выходные данные снова в команду `cut`, используя параметр `-d` (delimiter — разделитель), чтобы изменить символ-разделитель на запятую вместо табуляции. Это позволит выделить только фамилии авторов:

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

Экономьте время благодаря истории команд и редактированию

Приходилось ли вам набирать одни и те же команды заново? Вместо этого несколько раз нажмите клавишу со стрелкой вверх, чтобы просмотреть команды, которые вы запускали ранее (эта функция оболочки называется *историей команд*). Когда дойдете до нужной команды, нажмите Enter, чтобы немедленно запустить ее, или сначала отредактируйте, используя клавиши со стрелками влево и вправо для позиционирования курсора и клавишу Backspace для удаления (эта функция называется *редактирование командной строки*). В главе 3 мы узнаем о значительно более мощных функциях истории команд и редактирования.

Команда #4: grep

`grep` — чрезвычайно мощная команда, но пока скроем большую часть ее возможностей и ограничимся тем, что она печатает строки, соответствующие заданному шаблону (более подробная информация будет представлена в главе 5). Например, следующая команда отображает строки из файла *animals.txt*, содержащие текст `Nutshell`:

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

Также можно вывести строки, которые не соответствуют заданному шаблону, с опцией `-v`. Обратите внимание, что строки, содержащие `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
```

Таким образом, команда `grep` полезна для поиска определенного текста в некотором списке файлов. Следующая команда печатает строки, содержащие текст `Perl`, в файлах с расширением *.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 данном случае команда `grep` нашла три соответствия — в одной строке файла *animals.txt* и в двух строках файла *essay.txt*.

`grep` читает стандартный ввод и записывает стандартный вывод, что делает эту команду идеальной для конвейеров. Допустим, мы хотим узнать, сколько подкаталогов находится в каталоге */usr/lib*. Нет простой команды Linux для получения ответа, поэтому создадим конвейер. Начнем с команды `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
⋮
```

Обратите внимание, что `ls -l` помечает каталоги буквой `d` в начале строки. Используем `cut`, чтобы вывести первый столбец:

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

Затем используем `grep`, чтобы оставить только строки, содержащие букву `d`:

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

Наконец, подсчитаем строки с помощью команды `wc` — и мы получим ответ, созданный конвейером из четырех команд: */usr/lib* содержит 145 подкаталогов:

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

Команда #5: sort

Команда `sort` сортирует строки файла в порядке возрастания (по умолчанию):

```
$ 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 убывания (с параметром `-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 сортировать строки в алфавитном порядке (по умолчанию) или в числовом порядке (с опцией `-n`). Продемонстрируем это на примере конвейеров, которые вырезают третье поле (год публикации) 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
```

а 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.

а затем отсортировать их:

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

Создадим сортированный список всех имен пользователей, а не только первых пяти из них, заменив `head` на `cat`:

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

Чтобы определить, есть ли у данного пользователя учетная запись в системе, сопоставим его логин с помощью `grep`. Пустой вывод означает отсутствие учетной записи:

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

Параметр `-w` указывает команде `grep` сопоставлять только слова целиком, а не их части. Если в системе имеется имя пользователя, содержащее *jones*, например `sallyjones2`, то команда из примера выше его не отобразит.

Команда #6: uniq

Команда `uniq` обнаруживает повторяющиеся соседние строки в файле. По умолчанию она удаляет повторы. Продемонстрируем это на файле, содержащем только заглавные буквы:

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

Обратите внимание, что команда `uniq` сократила первые три строки A до одной A, но оставила последнюю A на месте, потому что она не была *соседней* с первыми тремя.

С помощью параметра `-c` можно подсчитать количество повторяющихся строк:

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

Признаюсь честно, когда я впервые столкнулся с командой `uniq`, то не увидел в ней особой пользы, но она быстро стала одной из моих любимых. Предположим, у нас есть разделенный табуляцией файл с итоговыми оценками студентов университета в диапазоне от A (лучшая) до F (худшая):

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

Нам нужно вывести на экран оценку, которую получила бóльшая часть студентов (если будет равное количество у нескольких оценок, в вывод попадет первая из них). Начнем с извлечения столбца с оценками с помощью команды `cut`:

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

Затем используем команду `uniq` для подсчета совпадающих строк:

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

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

Затем отсортируем строки в порядке убывания, чтобы переместить наиболее часто встречающуюся оценку в верхнюю строку:

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

и оставим только первую строку с помощью команды `head`:

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

И наконец, поскольку нам нужна только буквенная оценка, а не количество, извлечем ее с помощью `cut`:

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

И вот вам ответ, благодаря конвейеру из шести команд — пока что самому длинному из использованных нами. Такого рода пошаговое построение конвейера — не просто обучающее упражнение. Именно так на самом деле работают эксперты Linux. Этой технике посвящена глава 8.

Обнаружение дубликатов файлов

Давайте объединим в большом примере все то, что уже узнали. Предположим, мы находимся в каталоге, который забит файлами JPEG, и хотим узнать, 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 на этот вопрос можно с помощью конвейера. Нам понадобится еще одна команда, `md5sum`, которая проверяет содержимое файла и вычисляет 32-символьную строку, называемую *контрольной суммой*:

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