9

Поиск данных

С каждым годом объем жестких дисков возрастает, а цена на них снижается. Появление новых технических средств — цифровых фотоаппаратов, видеокамер, mp3-плейеров — приводит к генерации большого объема данных, которыми заполняется дополнительное дисковое пространство. Увеличение объема доступной информации порождает новые проблемы: найти нужные данные оказывается все труднее. Непросто отыскать фотоснимок среди 10000 изображений или текстовый файл среди 600 других документов. К счастью, в системе Linux есть мощные средства поиска, позволяющие быстро и эффективно найти требуемую информацию.

Поиск в базе имен файлов

`locate`

Бывает ли у вас такое, что вы знаете имя файла или часть имени, но не представляете, в каком месте файловой системы он находится? Решить эту проблему вам поможет команда `locate`. Она ищет файлы и каталоги по заданному имени. Результаты поиска выводятся на терминал.

```
$ locate haggard
.../txt/rider_haggard
.../txt/rider_haggard/Queen_of_the_Dawn.txt
.../txt/rider_haggard/Allan_and_the_Ice-Gods.txt
.../txt/rider_haggard/Heu-Heu_or_The_Monster.txt
```

Результаты отображаются быстро, однако программа `locate` не выполняет реального поиска. Вместо этого она просматривает базу данных имен файлов, которая ежедневно автоматически обновляется (подробнее этот вопрос будет рассмотрен далее в данной главе). Поскольку `locate` ищет информацию в предварительно созданной базе, она выполняет поставленную перед ней задачу практически мгновенно.

На вашем компьютере, возможно, вместо `locate` установлена программа `slocate`. Команда `slocate` (secure locate) не предпринимает попыток поиска в тех каталогах, которые пользователь, вызвавший ее, не имеет права просматривать (например, если вы не являетесь пользователем `root`, вам недоступно содержимое каталога `/root`). До появления `slocate` программа `locate` генерировала множество ошибок, связанных с правами доступа. С началом использования `slocate` эти ошибки — дело прошлого.

Особенности работы `slocate` видны на следующем примере:

```
$ locate slocate.db
$ su -
# locate slocate.db
/var/lib/slocate/slocate.db.tmp
/var/lib/slocate/slocate.db
```

Поскольку сначала вы были зарегистрированы как обычный пользователь (не `root`), первая попытка поиска окончилась неудачей. Используя команду `su`, вы переключились для работы от имени `root`, а затем снова запусти-

ли locate. Теперь поиск дал результаты (slocate.db — это база данных, используемая slocate).

Для удобства пользователей во многих системах создают ссылку /usr/bin/locate, которая реально указывает на /usr/bin/slocate. Для того чтобы выяснить, сделано ли это в вашей системе, попробуйте выполнить следующую команду (в данном примере приведены результаты, полученные в K/Ubuntu 5.10):

```
$ ls -l /usr/bin/locate
root  root  /usr/bin/locate -> slocate
```

Поскольку пользователь может и не знать, что вызывает именно slocate, а также потому, что в имени locate на одну букву меньше и ее быстрее вводить, мы будем в данной книге говорить о locate, не задумываясь о том, какая программа выполняется реально.

Поиск в базе имен файлов без учета регистра

locate -i

В предыдущем примере мы искали файлы или каталоги, содержащие в своем имени последовательность символов haggard. Полученные результаты выглядели следующим образом:

```
$ locate haggard
.../txt/rider_haggard
.../txt/rider_haggard/Queen_of_the_Dawn.txt
.../txt/rider_haggard/Allan_and_the_Ice-Gods.txt
.../txt/rider_haggard/Heu-Heu_or_The_Monster.txt
```

Они были получены потому, что именно этот фрагмент текста составлял часть имени каталога. Но если бы каталог назывался H_Rider_Haggard, поиск не дал бы результатов. Когда мы задаем опцию -i, при поиске не учитывается

регистр символов. В этом случае были бы найдены файлы, в названии пути к которым содержалось бы haggard или Haggard (а также HAGGARD и даже HaGgArD).

```
$ locate -i haggard
/txt/rider_haggard
/txt/rider_haggard/Queen_of_the_Dawn.txt
/txt/rider_haggard/Allan_and_the_Ice-Gods.txt
/txt/rider_haggard/Heu-Heu_or_The_Monster.txt
/txt/Rider_Haggard
/txt/Rider_Haggard/King_Solomons_Mines.txt
/txt/Rider_Haggard/Allan_Quatermain.txt
```

Оказывается, в системе есть два каталога, удовлетворяющие новому критерию поиска. Если, вызывая locate, вы хотите получить максимальное количество результатов, не забывайте задавать опцию -i, в противном случае вы можете пропустить именно те файлы или каталоги, которые ищете.

Управление результатами поиска в базе имен файлов

-n

Если вы часто используете команду locate, то наверняка попадали в ситуации, подобные следующей:

```
$ locate pdf
/etc/cups/pdftops.conf
/etc/xpdf
/etc/xpdf/xpdfrc-latin2
```

В этом примере информация усечена. На моем компьютере такой поиск дал 2373 результата. Это слишком много! Чтобы просмотреть их, лучше использовать такую конструкцию:

```
$ locate pdf | less
```

В данном случае вы организуете конвейерную обработку, используя результаты поиска с помощью `locate` в качестве входных данных программы `less` (вопросы передачи данных для обработки команде `less` рассматривались в главе 5). Теперь 2373 строки результатов можно просмотреть постранично.

Если вам нужны лишь первые `x` результатов, используйте опцию `-n`, указав после нее число требуемых результатов.

```
$ locate -n 3 pdf
/etc/cups/pdftops.conf
/etc/xpdf
/etc/xpdf/xpdfrc-latin2
```

Теперь объем информации достаточно мал, и не исключено, что это все, что вам надо. Не позволяйте `locate` генерировать большой объем данных; управляйте ее выходом в соответствии со своими целями.

Обновление базы, используемой программой locate

updatedb

В первом разделе этой главы, где лишь начинался разговор о `locate`, было упомянуто, что причина столь быстрой работы команды в том, что поиск реально выполняется в базе данных, содержащей имена файлов и каталогов. При инсталляции программа `locate` автоматически настраивается на просмотр жесткого диска и обновление базы. Обычно такое обновление происходит ночью. Это очень удобно, но такой подход не позволяет найти файлы, которые лишь недавно были помещены в файловую систему.

Предположим, например, что вы инсталлировали Rootkit Hunter (программу, которая выявляет средства, используемые злоумышленниками), а потом хотите получить информацию об установленных файлах. Команда `locate` не поможет вам, поскольку она ничего не знает об этих файлах и не будет знать о них до тех пор, пока база данных не обновится. При необходимости вы можете в любое время вручную задать обновление базы, используемой `locate`. Для этого надо вызвать команду `updatedb`. Поскольку данная команда индексирует практически каждый файл и каталог на вашем компьютере, для вызова ее вам необходимы права `root` (в системах типа K/Ubuntu можно также задать `sudo`).

```
# apt-get install rkhunter
# exit
$ locate rkhunter
$ su -
# updatedb
# exit
$ locate rkhunter
/usr/local/rkhunter
/usr/local/rkhunter/bin
/usr/local/rkhunter/etc
```

В предыдущем примере мы сначала установили `rkhunter` (пакет Rootkit Hunter), а затем завершили сеанс пользователя `root`. После этого был выполнен поиск `rkhunter`, но он не дал результатов. Мы снова выступили под именем `root`, запустили `updatedb` для просмотра жесткого диска, в результате чего `locate` получила информацию об изменениях, а затем завершили сеанс `root`. И наконец, мы снова выполнили поиск `rkhunter` с помощью программы `locate` и на этот раз он был успешным.

Запуская updatedb, необходимо учитывать следующее: время работы этой программы прямо пропорционально числу файлов на вашем жестком диске и быстродействию компьютера. Если у вас быстрый процессор, быстрый диск и немного файлов, updatedb завершится быстро. А что если быстродействие процессора мало, скорость обмена с жестким диском невелика, а на диске миллионы файлов? Тогда придется запастись терпением. Если вы хотите узнать, сколько времени потребовалось для индексирования содержимого файловой системы, задайте перед updatedb команду time так, как показано ниже.

```
# time updatedb
```

Когда updatedb завершит работу, time сообщит о затраченном времени. Эти сведения будут полезны на случай, если вы не обладаете достаточным запасом времени и вам надо решить, имеет ли смысл вызывать updatedb.

На заметку

Команда updatedb дает те же результаты, что и запуск slocate с опцией -u. Более того, как нетрудно выяснить, updatedb — это всего лишь ссылка на slocate.

```
$ ls -l /usr/bin/updatedb
root root /usr/bin/updatedb -> slocate
```

Поиск фрагментов текстового файла

grep

Команда locate позволяет организовать поиск имен файлов и каталогов, но не дает возможности искать информацию в составе файлов. Для того чтобы решить эту задачу, надо использовать команду grep. При работе с этой командой ей задается шаблон поиска и файл или группа файлов

(или даже весь жесткий диск), в которых надо найти фрагмент, соответствующий шаблону. В результате выполнения grep отображает строки, в которых присутствует интересующий вас фрагмент.

```
$ grep pain three_no_more_forever.txt
all alone and in pain
```

В данном случае мы используем команду grep для проверки, содержится ли в указанном файле слово pain. Как видите, результат проверки положительный; grep выводит на экран строку, содержащую данное слово. А можно ли выполнить поиск в нескольких файлах? Сделать это позволяют символы групповых операций.

```
$ grep pain *
fiery inferno in space.txt:watch the paint peel,
three_no_more_forever.txt:all alone and in pain
the speed of morning.txt:of a Chinese painting.
8 hour a day.txt:nice paint job too
ghost pain.txt:Subject: ghost pain
```

Заметьте, что команда grep нашла все вхождения ключевого слова pain, в том числе paint и painting. Также обратите внимание на то, что данная команда выводит не только строки, содержащие искомое слово, но и имена файлов. До сих пор мы не испытывали трудностей, выполняя поиск посредством grep. Пора усложнить задачу.

Общие сведения о шаблонах поиска

Из предыдущего раздела вы узнали, что программа grep позволяет находить указанное слово в группе файлов. Это одно из самых простых применений grep. Теперь попробуем глубже разобраться в структуре шаблонов, используемых для поиска. При формировании этих шаблонов при-

меняется один из самых мощных инструментов Linux: регулярные выражения. Для того чтобы в полной мере воспользоваться возможностями, предоставляемыми программой `grep`, надо изучить механизм регулярных выражений. Однако, для того, чтобы подробно рассмотреть этот вопрос, потребовалась бы отдельная книга, поэтому здесь мы обсудим лишь общие положения.

Совет

Если вы хотите получить дополнительную информацию о регулярных выражениях, ее легко найти в глобальной сети. Но на мой взгляд, самым лучшим пособием будет книга Бена Форты *Освой самостоятельно регулярные выражения. 10 минут на урок* (ИД "Вильямс", 2004 г.).

Одна из особенностей, затрудняющих начинающим пользователям изучение `grep`, состоит в том, что существует несколько версий данной команды. Сведения о них приведены в табл. 9.1.

Таблица 9.1. Различные версии программы `grep`

Интерпретация шаблона	Опция команды grep	Отдельная команда
Основные регулярные выражения	`grep -G` (или `--basic-regexp`)	`grep`
Расширенные регулярные выражения	`grep -E` (или `--extended-regexp`)	`egrep`
Набор фиксированных строк, для каждой из которых может быть установлено соответствие	`grep -F` (или `--fixed-strings`)	`fgrep`
Регулярные выражения Perl	`grep -P` (или `--perl-regexp`)	Отсутствует

Как видно из таблицы, сама по себе команда `grep` поддерживает основные регулярные выражения. Если вы зададите опцию `-E` (или `--extended-regexp`) либо команду `egrep`, то сможете использовать расширенные регулярные выражения. Именно такие выражения вам потребуются в большинстве случаев, за исключением разве что очень простых критериев поиска. Более сложные варианты команды — это `grep` с опцией `-F` (или `--fixed-strings`) или команда `fgrep`, позволяющая задавать множественные условия поиска, и `grep` с опцией `-P` (или `--perl-regexp`), которая дает возможность применять конструкции, типичные для языка Perl.

На заметку

В данной книге мы будем в основном применять обычную команду `grep`, поддерживающую основные регулярные выражения. Отклонения от этого правила будут оговариваться отдельно.

Перед тем как продолжить изучение материала, надо обсудить некоторые особенности, которые могут затруднить его восприятие.

Символы групповых операций и регулярные выражения — не одно и то же. Например, и в том и в другом случае используется символ *, однако он интерпретируется поразному. Если в групповых операциях знаки ? и * означают произвольные символы, то в групповых операциях они управляют повторением предшествующей им конструкции. Например, символ групповой операции ? в `c?t` заменяется одним и только одним символом. Этому выражению соответствуют слова `cat`, `cot` и `cut`, но не `ct`. В регулярных выражениях символ ? в конструкции `c[a-z]?t` указывает на то, что буква из диапазона `a-z` может присутствовать один раз или отсутствовать вовсе. Такому шаблону соответствуют последовательности символов `cat`, `cot`, `cut`, а также `ct`.

Совет

Чтобы лучше понять различия между символами групповых операций и регулярными выражениями, обратитесь к документам *What is a Regular Expression* (`http://docs.kde.org/stable/en/kdeutils/KRegExpEditor/whatIsARegExp.html`), *Regular Expressions Explained* (`http://www.castaglia.org/proftpd/doc/contrib/regexp.html`) и *Wildcards Gone Wild* (`http://www.linux-mag.com/2003-12/power_01.html`).

Источником проблем при работе с командой `grep` могут также стать символы, имеющие специальное назначение. Например, выражение `[a-e]` обозначает диапазон символов, ему соответствует только одна из следующих букв: `a`, `b`, `c`, `d` или `e`. Используя `[` или `]` в регулярном выражении, надо различать случаи, когда они обозначают границы диапазона или непосредственно входят в последовательность символов, предназначенную для поиска. Символы, имеющие специальное значение, приведены ниже.

```
. ? [ ] ^ $ | \
```

И наконец, одинарная и двойная кавычки в регулярных выражениях существенно отличаются друг от друга. Одинарная кавычка сообщает о том, что должна быть найдена строка символов, а двойная кавычка указывает на использование переменных оболочки. Рассмотрим строку `hey you!`, заданную в качестве условия поиска.

```
$ grep hey you! *
grep: you!: No such file or directory
txt/pvzm/8 hours a day.txt:hey you! let's run!
txt/pvzm/friends & family.txt:in patience they wait
txt/pvzm/speed of morning.txt:they say the force
```

Поскольку в данном случае последовательность символов `hey you` была включена в состав команды без ограничи-

телей, команда grep неправильно обработала запрос. Она интерпретировала первое слово, hey, как ключевое, а второе слово, you!, как имя файла. Такого файла не существует, поэтому поиск в нем не может быть выполнен. Затем, восприняв символ *, она стала искать слово hey в каждом файле, содержащемся в текущем каталоге, и вернула три строки результатов. В первой из этих строк содержится фраза, которую требовалось найти, но сказать о том, что поиск был выполнен корректно, нельзя. Повторим попытку.

Ограничим последовательность символов для поиска двойными кавычками. Будет ли устранена проблема, возникшая в предыдущем случае?

```
$ grep "hey you!" *
bash: !" *: event not found
```

Стало еще хуже! Включив в выражение двойные кавычки, мы допустили большую ошибку и в результате не получили даже тех строк, которые команда grep нашла в прошлый раз. Что случилось? Символ ! — это команда оболочки, ссылающаяся на список предыстории. По правилам за ним должен следователь идентификатор процесса, представляющий команду, выполненную ранее, например !264. Таким образом, bash ожидает идентификационный номер после символа ! и сообщает, что не может найти команду " * (двойная кавычка, пробел и звездочка).

Дело в том, что двойная кавычка указывает на использование в шаблоне поиска переменных оболочки, хотя мы совсем не собирались применять их. Таким образом, двойные кавычки в данном случае неуместны. Попробуем использовать одинарные кавычки.

```
$ grep 'hey!' *
txt/pvzm/8 hours a day.txt:hey you! let's run!
```

Теперь все намного лучше. Одинарные кавычки сообщают grep не о переменных оболочки, а лишь о том, что критерием поиска является обычная строка символов. Результатом является строка, содержащая указанное выражение, чего мы и добивались.

Какие можно сделать выводы? Необходимо знать, когда использовать одинарные кавычки, когда двойные, а когда можно обойтись без них. Если вы ищете конкретную последовательность символов, ее надо поместить в одинарные кавычки. Если же вы хотите включить в эту последовательность переменную оболочки (такая необходимость возникает крайне редко), следует использовать двойные кавычки. Если же вы ищете одно слово, содержащее лишь числа и буквы, вполне можно обойтись без кавычек. Желая перестраховаться, поместите слово в одинарные кавычки — вреда от этого не будет.

Рекурсивный поиск фрагментов текста в файлах

-R

Символ групповой операции * позволяет указать несколько файлов в одном каталоге, но для поиска в нескольких подкаталогах вам понадобится опция -R (или --recursive). Попробуем найти слово hideous, которое любили употреблять авторы романов ужасов в XIX и начале XX столетия. Поиск будем производить в наборе файлов, содержащем литературные произведения.

```
$ grep -R hideous *
machen/great_god_pan.txt:know, not in
your most fantastic, hideous dreams can you have
machen/great_god_pan.txt:hideously
contorted in the entire course of my practice, and
```

```
machen/great_god_pan.txt:death was
horrible. The blackened face, the hideous form upon
lovecraft/Beyond the Wall of Sleep.txt:some
hideous but unnamed wrong, which
lovecraft/Beyond the Wall of Sleep.txt:blanket
over the hideous face, and awakened the nurse.
lovecraft/Call of Cthulhu.txt:hideous a chain. I
think that the professor, too, intended to
lovecraft/Call of Cthulhu.txt:voodoo meeting;
and so singular and hideous were the rites
lovecraft/Call of Cthulhu.txt:stated, a very
crude bas-relief of stone, comprising a hideous...
```

Совет

Если вы получите слишком много результатов, вы можете передать их средствами конвейерной обработки программе `less`.

```
$ grep -R hideous * | less
```

Можно также перенаправить вывод команды в текстовый файл, а потом просмотреть его содержимое с помощью текстового редактора.

```
$ grep -R hideous * > hideous_in_horror.txt
```

Этот способ хорош тем, что вы сможете в любой момент вернуться к полученным результатам, не выполняя снова команду `grep`.

Поиск фрагментов текста в файлах без учета регистра

```
-i
```

По умолчанию при поиске, осуществляемом посредством команды `grep`, учитывается регистр символов. В предыдущем разделе мы искали слово `hideous`. А как насчет `Hideous`?

```
$ grep Hideous h_p_lovecraft/*
h_p_lovecraft/the_whisperer_in_darkness.txt: them.
Hideous though the idea was, I knew...
```

Поиск по слову `hideous` дал 463 результата, а по слову `Hideous` — один. Можно ли задать поиск обоих слов? Это позволяет опция `-i` (или `--ignore-case`), которая указывает, что при поиске не должен учитываться регистр, поэтому будут найдены не только `hideous` и `Hideous`, но также `HiDeOuS` и `HIDEOUS` и другие сочетания строчных и прописных букв, если они, конечно, будут встречаться в тексте.

```
$ grep -i hideous h_p_lovecraft/*
h_p_lovecraft/Call of Cthulhu.txt:voodoo meeting;
and so singular and hideous were the rites
h_p_lovecraft/Call of Cthulhu.txt:stated, a very
crude bas-relief of stone, comprising a hideous
h_p_lovecraft/the_whisperer_in_darkness.txt: them.
Hideous though the idea was, I knew...
```

При этом надо учитывать, что в общем случае при использовании опции `-i` число результатов может увеличиться в несколько раз и даже на порядки. В конце предыдущего раздела были приведены рекомендации, как справиться со слишком большим объемом информации, возвращаемой командой `grep`.

Поиск слов в файлах

`-w`

Вспомним раздел, в котором мы только начали изучать команду `grep`. Мы искали слово `pain`, а команда `grep` вернула нам список строк, в которых встречалась данная последовательность символов.

```
$ grep pain *
fiery inferno in space.txt:watch the paint peel,
three_no_more_forever.txt:all alone and in pain
the speed of morning.txt:of a Chinese painting.
8 hour a day.txt:nice paint job too
ghost pain.txt:Subject: ghost pain
```

По умолчанию grep находит все вхождения указанной строки, в данном случае pain, в том числе слова paint и painting. Если бы в тексте присутствовали слова painless, Spain или painstaking, они также были бы учтены. Но что делать, если необходимы только те строки, в которые входит именно слово pain? Для этой цели предназначена опция -w (или --word-regexp).

```
$ grep -w pain *
three_no_more_forever.txt:all alone and in pain
ghost pain.txt:Subject: ghost pain
```

Данная опция позволяет сузить поиск и соответственно уменьшить набор результатов.

Отображение номеров строк

-n

Команда grep отображает каждую строку, содержащую заданную последовательность символов, но не сообщает о том, в какой части файла расположена эта строка. Чтобы получить номер строки, надо использовать опцию -n (или --line-number).

```
$ grep -n pain *
fiery inferno in space.txt:56:watch the paint peel,
three_no_more_forever.txt:19:all alone and in pain
the speed of morning.txt:66:of a Chinese painting.
8 hour a day.txt:78:nice paint job too
ghost pain.txt:32:Subject: ghost pain
```

Теперь, когда мы знаем номера строк, содержащих последовательность символов `pain`, достаточно просто, используя любой текстовый редактор, обратиться непосредственно к нужной строке.

Поиск слов в выходных данных других команд

```
$ ls -1 | grep 1960
```

Команда `grep` сама по себе является мощным средством поиска, но ее также можно использовать как фильтр для обработки выходных данных, сгенерированных другими программами. Предположим, например, что вы храните mp3-файлы с записями Джона Колтрейна, выделив для каждого альбома отдельный подкаталог, причем имя подкаталога начинается с года, в котором соответствующий альбом был выпущен. Часть информации, полученной с помощью команды `ls -1`, приведена ниже (опция `-1` приводит к тому, что в каждой строке отображается только одно имя).

```
$ ls -1
1956_Coltrane_For_Lovers
1957_Blue_Train
1957_Coltrane_[Prestige]
1957_Lush_Life
1957_Thelonious_Monk_With_John_Coltrane
```

Предположим теперь, что вам необходимо получить список альбомов, выпущенных в 1960 году. Решить эту задачу можно, передав результаты выполнения `ls -1` команде `grep`.

```
$ ls -1 | grep 1960
1960_Coltrane_Plays_The_Blues
1960_Coltrane's_Sound
```

```
1960_Giant_Steps
1960_My_Favorite_Things
```

Немного поразмыслив, вы придумаете сотни вариантов применения команды grep. Рассмотрим еще один пример. Команда ps выводит список выполняющихся процессов, причем опция -f указывает ps на то, что необходимо отобразить полный список с подробной информацией о каждом процессе, а опция -U, за которой следует пользовательское имя, ограничивает вывод лишь процессами, принадлежащими конкретному пользователю. Задав -fU scott, мы, вероятнее всего, получим длинный список, в котором очень сложно будет найти информацию о требуемом процессе. Команда grep позволяет сократить объем выходных данных.

На заметку

Для экономии места часть информации, обычно отображаемой по команде ps, здесь не приводится.

```
$ ps -fU scott | grep firefox
scott 17623 /bin/sh /opt/firefox/firefox
scott 17634 /opt/firefox/firefox-bin
scott  1601 grep firefox
```

Команда ps выводит информацию о всех процессах, принадлежащих пользователю scott (в данном случае их 64), но выходные данные посредством механизма конвейерной обработки передаются программе grep, отсеивающей все строки, в которых не содержится слово firefox. К сожалению, последняя строка лишняя: нам нужна информация о самой программе firefox, а не о ее имени, переданном команде grep. Скрыть эту строку можно следующим образом:

```
$ ps -fU scott | grep [f]irefox
scott 17623 /bin/sh /opt/firefox/firefox
scott 17634 /opt/firefox/firefox-bin
```

Теперь первая буква слова, передаваемого grep, лежит в диапазоне от f до f. Этому критерию соответствуют строки, сгенерированные программой ps, которые содержат слово firefox. Последовательность [f]irefox, переданная grep, не удовлетворяет условиям поиска. поскольку в ней содержатся символы [и]. Сама же команда grep использует шаблон [f]irefox, которому соответствует только слово firefox. Такой способ выглядит несколько сложно, но он дает нужные результаты. Возьмите его на вооружение!

Просмотр контекста для слов, имеющихся в файлах

-A, -B, -C

Когда речь идет о работе с данными, важность контекста трудно переоценить. Как вы знаете, программа grep выводит строку, содержащую заданную последовательность символов, но мы также можем указать grep выводить предыдущие и последующие строки. В предыдущем разделе grep использовалась для обработки списка альбомов Джона Колтрейна. Одним из наилучших считается "A Love Supreme". Какие три альбома были выпущены перед ними? Чтобы получить ответ, зададим опцию -B (или --before-context=#).

```
$ ls -1 | grep -B 3 A_Love_Supreme
1963_Impressions
1963_John_Coltrane_&_Johnny_Hartman
1963_Live_At_Birdland
1964_A_Love_Supreme
```

Если вас интересует, какие альбомы последовали за A Love Supreme, задайте опцию -A (или --after-context=#).

```
$ ls -1 | grep -A 3 'A_Love_Supreme'
1964_A_Love_Supreme
1964_Coltrane's_Sound
1964_Crescent
1965_Ascension
```

Для того чтобы получить полный контекст, можно использовать опцию -C (или --context=#), которая является сочетанием двух опций, рассмотренных выше.

```
$ ls -1 | grep -C 2 'A_Love_Supreme'
1963_John_Coltrane_&_Johnny_Hartman
1963_Live_At_Birdland
1964_A_Love_Supreme
1964_Coltrane's_Sound
1964_Crescent
```

Такая информация в некоторых случаях может быть сложной для восприятия, поскольку при обнаружении соответствия условиям поиска программа отображает сразу несколько строк. Так, например, в названиях ряда альбомов Колтрейна присутствует слово "live", и если вы зададите вывод предшествующих и последующих альбомов, результаты могут оказаться несколько неожиданными.

```
$ ls -1 | grep -C 1 Live
1963_John_Coltrane_&_Johnny_Hartman
1963_Live_At_Birdland
1964_A_Love_Supreme
--
1965_Last_Trane
1965_Live_in_Seattle
1965_Major_Works_of_John_Coltrane
--
```

```
1965_Transition
1966_Live_at_the_Village_Vanguard_Again!
1966_Live_in_Japan
1967_Expression
1967_Olatunji_Concert_Last_Live_Recording
1967_Stellar_Regions
```

Символы -- отделяют одну группу результатов от другой. Первые две группы очевидны — рядом с альбомом со словом “Live” в названии находятся другие альбомы, однако последняя группа выглядит гораздо сложнее. Несколько альбомов со словом “Live” непосредственно следуют друг за другом, поэтому результаты объединены. Это может показаться странными, но если вы ищете каждое вхождение слова “Live”, то заметите, что отображается информация о предшествующем и последующем альбомах.

Результаты станут еще более информативными, если мы дополнительно укажем при вызове команды опцию -n, которая задает вывод номеров строк.

```
$ ls -1 | grep -n -C 1 Live
37-1963_John_Coltrane_&_Johnny_Hartman
38:1963_Live_At_Birdland
39-1964_A_Love_Supreme
--
48-1965_Last_Trane
49:1965_Live_in_Seattle
50-1965_Major_Works_of_John_Coltrane
--
52-1965_Transition
53:1966_Live_at_the_Village_Vanguard_Again!
54:1966_Live_in_Japan
55-1967_Expression
56:1967_Olatunji_Concert_Last_Live_Recording
57-1967_Stellar_Regions
```

Теперь опция `-C` предоставляет еще больше информации о каждой строке: отображает после номера строки разные символы. Символ `:` указывает на то, что строка соответствует условиям поиска, а символ `-` означает, что это предшествующая или последующая строка. Строка `54:1966_Live_in_Japan` выполняет две функции: следует за строкой `53:1966_Live_at_the_Village_Vanguard_Again!`, по этому критерию после номера должен быть указан символ `-`, и сама содержит последовательность символов, заданную при вызове команды, т.е. в ней должно быть двоеточие. Поскольку соответствие критерию поиска является более важным признаком, предпочтение отдается символу `:`.

Отображение строк, не содержащих указанных слов

`-v`

С момента смерти Джона Колтрейна прошло около 40 лет, но его альбомы по-прежнему популярны. Знатоки искусства также высоко ценят творчество группы “Led Zeppelin”, которая выпустила девять альбомов; в названиях многих из них присутствует имя группы (вы скажете, что четвертый альбом вовсе не имеет названия, но большинство критиков называют его “Led Zeppelin IV”). Предположим, что вы хотите получить список каталогов, в которые поместили альбомы “Led Zeppelin”, но в названии которых не содержатся сами слова “Led Zeppelin”. Опция `-v` (или `--invert-match`) позволяет вам отобразить только те результаты, которые не соответствуют заданному шаблону.

```
$ ls -1
1969_Led_Zeppelin
1969_Led_Zeppelin_II
```

```
1970_Led_Zeppelin_III
1971_Led_Zeppelin_IV
1973_Houses_Of_The_Holy
1975_Physical_Graffiti
1976_Presence
1979_In_Through_The_Out_Door
1982_Coda
$ ls -1 | grep -v Led_Zeppelin
1973_Houses_Of_The_Holy
1975_Physical_Graffiti
1976_Presence
1979_In_Through_The_Out_Door
1982_Coda
```

Опция `-v` дает возможность сократить набор результатов и отобразить только те из них, которые вам действительно нужны. Вряд ли вам придется использовать данную опцию очень часто, но в некоторых случаях она может оказаться полезной.

Отображение списка файлов, содержащих указанное слово

`-l`

Как уже неоднократно говорилось, команда `grep` выводит строки, содержащие слово, заданное для поиска. Однако в некоторых случаях строки бывают излишни; нужны лишь имена файлов, в которых находится слово. В одном из предыдущих разделов мы искали вхождение слова `hideous`. Опция `-l` (или `--files-with-matches`) дает возможность получить список файлов (опция `-i`, указанная в той же строке, задает поиск без учета регистра символов).

```
$ grep -il hideous h_p_lovecraft/*
h_p_lovecraft/Call of Cthulhu.txt
h_p_lovecraft/From Beyond.txt
h_p_lovecraft/The Case of Charles Dexter Ward.txt
```

Подобные результаты в особенности полезны в сочетании с другими командами. Например, если вы хотите вывести на печать список файлов из конкретного каталога, в которых встречается слово `hideous`, можете объединить команды `grep` и `lpr` следующим образом:

```
$ grep -il hideous h_p_lovecraft/* | lpr
```

Помните, что данная команда выведет список имен файлов, а не их содержимое (если вы хотите распечатать сами тексты, вам придется использовать команду `cat`).

Поиск слов в результатах поиска

`grep | grep`

Предположим, что нам надо получить список альбомов Джона Колтрейна, выпущенных в последние три года его творческой биографии. Сделать это достаточно просто.

```
$ ls -1 | grep 196[6-7]
1966_Live_at_the_Village_Vanguard_Again!
1966_Live_in_Japan
1967_Expression
1967_Olatunji_Concert_Last_Live_Recording
1967_Stellar_Regions
```

Диапазон значений `[5-7]` ограничивает список годами 1965–1967; в противном случае его размеры были бы очень велики. Пока все хорошо, но допустим, что нам надо, чтобы в этот список не входили альбомы, в названии которых есть слово `Live` (мы вряд ли поступаем правильно, отка-

зываясь от данных произведений, но ведь это всего лишь пример)?

Ниже представлено выражение, которое решает эту задачу.

```
$ ls -1 | grep 196[6-7] | grep -v Live
1967_Expression
1967_Stellar_Regions
```

Опция `-v`, рассмотренная нами ранее в этой главе, отсеивает результаты, содержащие слово "Live", но в данном случае нас интересует не это. Главное для нас — тот факт, что мы получаем выходные данные команды `ls -1`, передаем их команде `grep 196[5-7]`, а затем новые результаты снова подвергаем фильтрации посредством второй команды `grep`, на этот раз вызванной с опцией `-v Live`. В конце концов мы получаем именно то, что нам надо: список альбомов Джона Колтрейна, выпущенных в 1965–1967 годах, в названии которых не содержится слово "Live". Вот вы и ознакомились с еще одним набором возможностей, доступным из командной строки Linux.

Выводы

В данной главе внимание было уделено двум командам, которые часто используются на практике: `locate` и `grep`. Несмотря на то что эти команды принадлежат одной группе, реализующей средства поиска информации на компьютере, они решают разные задачи. Команда `locate` выполняет поиск по именам файлов, используя для ускорения работы специальную базу данных, а команда `grep` просматривает в реальном времени содержимое файлов и извлекает фрагменты, удовлетворяющие условиям поиска.

И команда `locate`, и команда `grep` — удобные средства, но возможности поиска в файловой системе далеко не

исчерпываются ими. Следующая глава посвящена одной из самых мощных и гибких команд системы Linux, которая дополняет `locate` и `grep`, а также может работать совместно с ними. Это команда `find`. Переходите к следующей главе, и начнем знакомство с ней.