**17. Поиск файлов**

Блуждая по системе Linux, мы совершенно ясно увидели, что типичная Linux-система содержит множество файлов. В связи с этим возникает вопрос: как искать нужные файлы? Мы уже знаем, что файловая система в Linux организована в соответствии с определенными соглашениями, которые переходили из одного поколения Unix-подобных систем в другое, но огромное число файлов может порождать труднопреодолимую проблему.

В этой главе мы рассмотрим два инструмента для поиска файлов в системе:

• locate – выполняет поиск файлов по именам.

• find – выполняет поиск файлов в иерархии каталогов.

Мы также познакомимся с командой, часто используемой вместе с командами поиска файлов для обработки списков с результатами:

• xargs – конструирует команды на основе данных, полученных из стандартного ввода, и выполняет их.

Дополнительно в этой главе будет представлена пара команд, которые помогут нам в наших исследованиях:

• touch – изменяет времена, ассоциированные с файлом.

• stat – выводит статус файла или файловой системы.

**locate — простой способ поиска файлов**

Программа locate выполняет быстрый поиск в базе данных имен файлов и выводит все имена, соответствующие искомой строке. Допустим, к примеру, что нужно найти все программы с именами, начинающимися с zip. Поскольку требуется найти программы, можно предположить, что имя каталога с программами оканчивается на bin/. Соответственно можно попробовать выполнить поиск с помощью locate, как показано ниже:

[me@linuxbox ~]$ **locate bin/zip**

locate выполнит поиск в базе данных имен файлов и выведет все имена, содержащие строку bin/zip:

/usr/bin/zip

/usr/bin/zipcloak

/usr/bin/zipgrep

/usr/bin/zipinfo

/usr/bin/zipnote

/usr/bin/zipsplit

Если к результатам поиска предъявляются более строгие требования, команду locate можно объединить с другими инструментами, такими как grep, позволяющими осуществить более сложный поиск:

[me@linuxbox ~]$ **locate zip | grep bin**

/bin/bunzip2

/bin/bzip2

/bin/bzip2recover

/bin/gunzip

/bin/gzip

/usr/bin/funzip

/usr/bin/gpg-zip

/usr/bin/preunzip

/usr/bin/prezip

/usr/bin/prezip-bin

/usr/bin/unzip

/usr/bin/unzipsfx

/usr/bin/zip

/usr/bin/zipcloak

/usr/bin/zipgrep

/usr/bin/zipinfo

/usr/bin/zipnote

/usr/bin/zipsplit

Программа locate существует уже много лет, и за эти годы было создано несколько ее вариантов, получивших широкое распространение. Два из них, наиболее часто используемые в современных дистрибутивах Linux, — это slocate и mlocate, которые, впрочем, являются символическими ссылками, указывающими на locate. Разные версии locate имеют пересекающиеся множества параметров. Некоторые поддерживают поиск с использованием регулярных выражений (о которых рассказывается в главе 19) и групповые символы. Загляните на страницу справочного руководства (man) для locate, чтобы определить, какая версия установлена у вас.

**откуда берется база данных для locate?**

В некоторых дистрибутивах при попытке запустить locate сразу после установки она потерпит неудачу, но если попытаться использовать ее на следующий день, все, как ни странно будет работать как надо. Так в чем же проблема? База данных для locate создается другой программой с именем updatedb. Обычно она периодически запускается как *задание* *cron*; то есть она запускается системным планировщиком cron через регулярные интервалы времени. В большинстве систем, в состав которых входит locate, программа updatedb запускается один раз в сутки. Поскольку база данных не обновляется непрерывно, скорее всего, locate не находит самые свежие файлы. Чтобы решить эту проблему, запустите программу updatedb вручную от имени суперпользователя.

**find — сложный способ поиска файлов**

В отличие от программы locate, выполняющей поиск файлов по именам, программа find ищет файлы согласно заданным атрибутам в указанном каталоге (и во вложенных подкаталогах). Далее мы потратим время на исследование команды find, потому что она имеет множество интересных особенностей, с которыми мы неоднократно столкнемся, когда начнем знакомиться с приемами программирования в последующих главах.

В простейшем случае программе find можно передать одно или несколько имен каталогов для поиска. Например, с ее помощью можно получить список содержимого домашнего каталога:

[me@linuxbox ~]$ **find ~**

Для большинства активных пользователей она выдаст длинный список. Так как список выводится в стандартный вывод, его можно передать по конвейеру другим программам. Воспользуемся программой wc, чтобы подсчитать число файлов:

[me@linuxbox ~]$ **find ~ | wc -l**

47068

Ух ты, вот это мы поработали! Вся прелесть команды find в том, что ее можно использовать для поиска файлов, соответствующих определенным критериям. Она делает это посредством (что немного странно) применения *проверок*, *операций* и *параметров*. Сначала рассмотрим проверки.

**Проверки**

Допустим, мы хотим получить список каталогов. Для этого добавим в команду следующую проверку:

[me@linuxbox ~]$ **find ~ -type d | wc -l**

1695

Добавив проверку -type d, мы ограничились поиском только каталогов. Но точно так же можно ограничить поиск только обычными файлами:

[me@linuxbox ~]$ **find ~ -type f | wc -l**

38737

В табл. 17.1 перечислены проверки типов файлов,  наиболее часто используемых с командой find.

**Таблица 17.1. Проверки типов файлов в find**

|  |  |
| --- | --- |
| **Тип файлов** | **Описание** |
| b | Специальный файл блочного устройства |
| c | Специальный файл символьного устройства |
| d | Каталог |
| f | Обычный файл |
| l | Символическая ссылка |

Добавив дополнительные проверки, можно выполнять поиск файлов по размеру и имени. Давайте найдем все обычные файлы с именами, соответствующими шаблону *\*.JPG*, и имеющие размер больше 1 мегабайта:

[me@linuxbox ~]$ **find ~ -type f -name "\*.JPG" -size +1M | wc -l**

840

В этом примере мы добавили проверку -name с шаблоном имени файла. Обратите внимание, что шаблон заключен в кавычки, чтобы предотвратить подстановку имен файлов командной оболочкой. Далее мы добавили проверку -size со строкой +1M. Начальный символ «плюс» указывает, что требуется искать файлы, размер которых превышает указанное число. Начальный символ «минус» изменил бы значение строки на противоположное: «меньше указанного числа». Число без знака означает: «в точности соответствует значению». Буква M в конце определяет единицы измерения – мегабайты (Megabytes). В табл. 17.2 перечислены символы, которые можно использовать для обозначения единиц измерения.

Команда find поддерживает множество разнообразных проверок. В табл. 17.3 приводится краткое описание наиболее часто используемых из них. Обратите внимание, что в случаях, когда требуется числовой аргумент, допустимо использование формы записи с символами + и -, обсуждавшейся выше.

**Таблица 17.2. Единицы измерения, поддерживаемые командой find**

| **Символ** | **Единица измерения** |
| --- | --- |
| b | Блоки размером по 512 байт (используется по умолчанию, если иное не указано явно) |
| c | Байты |
| w | 2-байтные слова |
| k | Килобайты (Kilobytes, блоки по 1024 байт) |
| M | Мегабайты (Megabytes,  блоки по 1 048 576 байт) |
| G | Гигабайты (Gigabytes, блоки по 1 073 741 824 байт) |

**Таблица 17.3. Проверки, поддерживаемые командой find**

| **Проверка** | **Описание** |
| --- | --- |
| -cmin n | Соответствует файлам или каталогам, содержимое или атрибуты которых последний раз изменялись точно n минут назад. Чтобы выразить условие «менее n минут назад», используйте -n; чтобы выразить условие «более n минут назад», используйте +n |
| -cnewer имя | Соответствует файлам или каталогам, содержимое или атрибуты которых последний раз изменялись позже, чем у файла с указанным именем |
| -ctime n | Соответствует файлам или каталогам, содержимое или атрибуты (то есть разрешения) которых последний раз изменялись более чем n\*24 часа назад |
| -empty | Соответствует пустым файлам и каталогам |
| -group группа | Соответствует файлам или каталогам, принадлежащим указанной группе. Группа может задаваться именем или числовым идентификатором группы |
| -iname шаблон | Действует так же, как проверка -name, но различает регистр символов |
| -inum n | Соответствует файлам с номером индексного узла (inode) n. Эту проверку удобно использовать для поиска всех жестких ссылок на определенный индексный узел |
| -mmin n | Соответствует файлам или каталогам, содержимое которых последний раз изменялось n минут назад |
| -mtime n | Соответствует файлам или каталогам, содержимое которых последний раз изменялось n\*24 часов назад |
| -name шаблон | Соответствует файлам и каталогам, имена которых совпадают с указанным шаблоном |
| -newer имя | Соответствует файлам и каталогам, содержимое которых последний раз изменялось позже, чем у файла с указанным именем. Эта проверка может пригодиться в сценариях, выполняющих резервное копирование файлов. Каждый раз в процессе создания резервной копии можно обновлять файл (например, файл журнала) и затем с помощью find определять, какие файлы изменились с момента последнего обновления |
| -nouser | Соответствует файлам и каталогам, не принадлежащим какому-либо допустимому пользователю. Эту проверку можно использовать для поиска файлов, принадлежащих удаленным учетным записям, или для обнаружения следов злоумышленников |
| -nogroup | Соответствует файлам и каталогам, не принадлежащим какой-либо допустимой группе |
| -perm режим | Соответствует файлам или каталогам с указанным режимом доступа. Режим может выражаться восьмеричным числом или иметь символическую форму |
| -samefile имя | Действует так же, как проверка -inum. Соответствует файлам с тем же номером индексного узла (inode), что и файл с указанным именем |
| -size n | Соответствует файлам с размером n |
| -type c | Соответствует файлам с типом c |
| -user имя | Соответствует файлам или каталогам, принадлежащим пользователю с указанным именем. Аргумент имя может быть именем или числовым идентификатором пользователя |

Это не полный список. Дополнительные детали ищите на странице справочного руководства (man) для команды find.

**Операторы**

Несмотря на большое число проверок, поддерживаемых командой find, мы все еще нуждаемся в способе определения *логических отношений* между проверками. Например, представьте, что в некотором каталоге мы хотим найти все файлы и подкаталоги с небезопасными разрешениями. Для этого можно было бы выполнить поиск всех файлов с разрешениями, отличающимися от 0600, и каталогов с разрешениями, отличающимися от 0700. К счастью, find поддерживает возможность комбинирования проверок с помощью логических операторов, с целью определить более сложные критерии отбора. Выразить вышеупомянутую проверку можно так:

[me@linuxbox ~]$ **find ~ \( -type f -not -perm 0600 \) -or \( -type d -not –perm 0700 \)**

Ф-фу! Как неизящно! Что все это значит? На самом деле операторы перестанут казаться избыточно сложными, как только вы с ними познакомитесь поближе (табл. 17.4).

Имея список операторов под рукой, попробуем разобрать команду find. На самом верхнем уровне мы видим, что проверки объединены в две группы, разделенные оператором -or:

(*выражение 1*) -or (*выражение 2*)

**Таблица 17.4. Логические операторы, поддерживаемые командой find**

|  |  |
| --- | --- |
| **Оператор** | **Описание** |
| -and | Соответствует, если выполняются условия в проверках с обеих сторон от оператора. Можно сократить до -a. Обратите внимание, что в отсутствие операторов по умолчанию подразумевается -and |
| -or | Соответствует, если выполняется условие с одной из сторон от оператора. Можно сократить до -o |
| -not | Соответствует, если условие в проверке, следующей за оператором, не выполняется. Можно сократить до -! |
| ( ) | Группируют проверки и операторы для формирования крупных выражений. Используются для управления порядком проверок. По умолчанию проверки выполняются слева направо. Часто используются для изменения порядка проверок по умолчанию, чтобы получить желаемый результат. Даже если скобки не нужны, иногда полезно включать их, чтобы сделать команды более наглядными. Не забывайте, что круглые скобки имеют специальное значение для командной оболочки, поэтому их нужно экранировать, чтобы они передавались команде find как аргументы. Обычно экранирование выполняют с помощью символа обратного слеша |

В этом есть определенный смысл, потому что мы хотим найти файлы с одним набором разрешений и каталоги – с другим. Но если выполняется поиск и файлов и каталогов, почему используется оператор -or вместо -and? Потому что find, выполняя обход файлов и каталогов, оценивает их по одному, чтобы понять, соответствует ли файл или каталог указанным проверкам. Команде требуется узнать, является ли очередной элемент файлом или каталогом с «плохими» разрешениями. Один и тот же элемент не может соответствовать сразу двум условиям. То есть если развернуть сгруппированные выражения, можно увидеть следующее:

(файл с плохими разрешениями) -or (каталог с плохими разрешениями)

Наша следующая задача — проверить «плохие разрешения». Как это сделать? Фактически никак. Но мы можем проверить «неудовлетворительные разрешения», зная, что такое «удовлетворительные разрешения». В случае с файлами удовлетворительными являются разрешения 0600, для каталогов — 0700. Выражение, проверяющее «неудовлетворительные» разрешения, выглядит так:

-type f -and -not -perms 0600

а для каталогов так:

-type d -and -not -perms 0700

Как отмечалось в табл. 17.4, оператор -and можно просто удалить, так как он подразумевается по умолчанию. Теперь, объединив все вместе, мы получим окончательную команду:

**find ~ (-type f -not -perms 0600) -or (-type d -not -perms 0700)**

Однако поскольку круглые скобки имеют специальное значение для командной оболочки, их нужно экранировать, чтобы предотвратить интерпретацию скобок командной оболочкой. Для этого достаточно добавить обратный слеш перед каждой из них.

Логические операторы имеют еще одну важную особенность, с которой необходимо разобраться. Представьте, что у нас есть два выражения, разделенных логическим оператором:

*выражение1*-оператор*выражение2*

*Выражение1* будет вычислено в любом случае, а вот будет ли вычислено *выражение2*, зависит от оператора. В табл. 17.5 показано, как это работает.

**Таблица 17.5. Действие логических операторов -and/-or команды find**

|  |  |  |
| --- | --- | --- |
| **Результат выражения1** | **Оператор** | **Выражение2...** |
| Истина | -and | Всегда вычисляется |
| Ложь | -and | Никогда не вычисляется |
| Истина | -or | Никогда не вычисляется |
| Ложь | -or | Всегда вычисляется |

Почему так происходит? Это сделано для повышения производительности. Возьмем для примера оператор -and. Мы знаем, что выражение выражение1 -and выражение2 не может быть истинным, если выражение1 вернет ложный результат, поэтому нет смысла вычислять выражение2. Аналогично, если имеется выражение выражение1 -or выражение2 и выражение1 вернет истинный результат, нет смысла вычислять выражение2, так как уже известно, что выражение1 -or выражение2 является истинным.

Это удобно, поскольку такой порядок вычислений помогает повысить скорость выполнения. Но почему это так важно для нас? Потому что мы можем использовать данную особенность для управления выполнением операций, о которых рассказывается далее.

**Операции**

Давайте попробуем выполнить определенные действия в процессе поиска! Иметь список с результатами работы команды find уже неплохо, но представьте, что нам нужно выполнить некие операции с элементами списка. К счастью, find позволяет выполнять наши операции, основываясь на результатах поиска.

**Предопределенные операции**

Существует множество предопределенных операций и несколько способов применения операций, определяемых пользователем. Для начала взгляните на неполный список предопределенных операций в табл. 17.6.

**Таблица 17.6. Предопределенные операции, поддерживаемые командой find**

|  |  |
| --- | --- |
| **Операция** | **Описание** |
| -delete | Удаляет текущий найденный файл |
| -ls | Действует эквивалентно команде ls -dils в отношении найденного файла. Результат выводится в стандартный вывод |
| -print | Выводит полный путь к найденному файлу в стандартный вывод. Эта операция выполняется по умолчанию, если не указана никакая другая |
| -quit | Завершает выполнение команды после обнаружения первого совпадения |

Поддерживаемых операций намного больше, чем показано здесь. Полный список можно найти на странице справочного руководства (man) для команды find.

В нашем первом примере мы выполнили команду:

**find ~**

Она выводит список всех файлов и подкаталогов, хранящихся в домашнем каталоге. Список выводится просто потому, что в отсутствие других операций предполагается операция -print. То есть эту команду можно было бы выразить так:

**find ~ -print**

Программу find можно использовать для удаления файлов, соответствующих определенным критериям. Например, следующая команда удалит все файлы с расширением *.BAK* (которое часто используется для обозначений резервных копий файлов):

**find ~ -type f -name '\*.BAK' -delete**

Эта команда найдет в домашнем каталоге (и во вложенных подкаталогах) пользователя все файлы с расширением *.BAK* и удалит их.

**ВНИМАНИЕ**

Обратите внимание, что операцию -delete следует использовать **с особыми предосторожностями**. Всегда предварительно проверяйте команду, подставив операцию -print вместо -delete, чтобы убедиться, что она не удалит ничего лишнего.

Прежде чем продолжить, давайте посмотрим, как логические операторы воздействуют на операции. Взгляните на следующую команду:

**find ~ -type f -name '\*.BAK' -print**

Как видите, эта команда ищет обычные файлы (-type f) с расширением *.BAK* (-name '\*.BAK') и выводит относительные пути к ним в стандартный вывод (-print). Однако такой порядок работы команды определяется логическими отношениями между всеми проверками и операциями. Как вы помните, между проверками и операциями по умолчанию подразумевается отношение -and. Ту же коман­ду можно  выразить, добавив логические операторы:

**find ~ -type f -and -name '\*.BAK' -and -print**

Теперь, имея перед глазами это определение, взгляните на табл. 17.7, где показано, как логические операторы влияют на порядок выполнения.

**Таблица 17.7. Влияние логических операторов**

|  |  |
| --- | --- |
| **Проверка/операция** | **Выполняется, когда...** |
| -print | -type f and -name '\*.BAK' истинно |
| -name '\*.BAK' | -type f истинно |
| -type f | Всегда выполняется, потому что это первая проверка/операция в отношении -and |

Так как логические отношения между проверками и операциями определяют необходимость их выполнения, можно сделать вывод, что их порядок следования играет важную роль. Например, если изменить порядок выполнения операций и проверок, поставив операцию -print на первое место, команда будет вести себя иначе:

**find ~ -print -and -type f -and -name '\*.BAK'**

Эта версия команды выведет каждый файл (операция -print всегда возвращает истинное значение), а затем проверит тип файла и его расширение.

**Операции, определяемые пользователем**

Помимо предопределенных операций можно также вызывать произвольные ­команды. Традиционно с этой целью используется операция -exec, что показано ниже:

-exec *команда* {} ;

где команда — это имя команды, {} — символическое представление текущего пути к файлу и точка с запятой — обязательный разделитель, обозначающий конец коман­ды. Следующий пример демонстрирует использование -exec для получения эффекта, аналогичного операции -delete, обсуждавшейся выше:

-exec rm '{}' ';'

И снова, поскольку фигурные скобки и точка с запятой имеют специальное значение для командной оболочки, они должны заключаться в кавычки или экранироваться.

Кроме того, существует возможность выполнять пользовательские операции интерактивно. Если заменить операцию -exec операцией -ok, перед выполнением каждой указанной команды будет выводиться запрос:

**find ~ -type f -name 'foo\*' -ok ls -l '{}' ';'**

< ls ... /home/me/bin/foo > ? **y**

-rwxr-xr-x 1 me   me 224 2011-10-29 18:44 /home/me/bin/foo

< ls ... /home/me/foo.txt > ? **y**

-rw-r--r-- 1 me   me   0 2012-09-19 12:53 /home/me/foo.txt

Эта команда ищет файлы с именами, начинающимися со строки foo, и для каждого найденного файла выполняет команду ls -l. Операция -ok запрашивает подтверждение у пользователя, прежде чем выполнить команду ls.

**Увеличение эффективности**

Каждый раз, когда обнаруживается файл, соответствующий критериям, операция -exec запускает новый экземпляр указанной команды. Но иногда желательно объединить все результаты поиска и запустить единственный экземпляр команды. Например, вместо последовательности команд, такой как:

ls -l *файл1*

ls -l *файл2*

предпочтительнее было бы выполнить команду:

ls -l *файл1* *файл2*

Здесь команда выполняется только один раз, а не несколько. Существует два способа добиться этого: традиционный, с использованием внешней команды xargs, и альтернативный, с использованием новой возможности в самой команде find. Обсудим сначала альтернативный способ.

Если заменить завершающий символ точки с запятой знаком «плюс», в команде find активируется функция объединения результатов в список аргументов для вызова единственного экземпляра требуемой команды. Вернемся к нашему примеру. Команда:

**find ~ -type f -name 'foo\*' -exec ls -l '{}' ';'**

-rwxr-xr-x 1 me   me 224 2011-10-29 18:44 /home/me/bin/foo

-rw-r--r-- 1 me   me   0 2012-09-19 12:53 /home/me/foo.txt

будет вызывать ls для каждого найденного файла. Изменив команду, как показано ниже:

**find ~ -type f -name 'foo\*' -exec ls -l '{}' +**

-rwxr-xr-x 1 me   me 224 2011-10-29 18:44 /home/me/bin/foo

-rw-r--r-- 1 me   me   0 2012-09-19 12:53 /home/me/foo.txt

мы получим тот же результат, но система выполнит команду ls только один раз.

Тот же результат можно получить с помощью команды xargs. Она принимает входные данные со стандартного ввода и преобразует их в список аргументов для указанной команды. В данном примере ее можно было бы использовать так:

**find ~ -type f -name 'foo\*' -print | xargs ls -l**

-rwxr-xr-x 1 me   me 224 2011-10-29 18:44 /home/me/bin/foo

-rw-r--r-- 1 me   me   0 2012-09-19 12:53 /home/me/foo.txt

Здесь вывод команды find передается по конвейеру команде xargs, которая, в свою очередь, конструирует список аргументов для команды ls и выполняет ее.

**ПРИМЕЧАНИЕ**

Несмотря на то что в командную строку можно включить большое число аргументов, оно не бесконечно. Не исключена возможность конструирования такой команды, которая окажется слишком велика для командной оболочки. Когда длина командной строки превышает максимально допустимый размер, xargs выполнит указанную команду с максимально возможным числом аргументов и затем повторит процесс, пока не исчерпает все, что получит со стандартного ввода. Чтобы увидеть максимально возможную длину командной строки, выполните xargs с параметром --show-limits.

**обработка файлов с необычными именами**

Unix-подобные системы позволяют встраивать в имена файлов пробелы (и даже символы перевода строки). Это порождает проблемы при выполнении программ, таких как xargs, конструирующих списки аргументов для других программ. Внутренние пробелы интерпретируются как разделители, и получившаяся команда будет интерпретировать слова, разделенные пробелами, как отдельные аргументы. Для решения этой проблемы find и xarg предлагают использовать в качестве разделителя аргументов *пустой символ* (null character). В кодировке ASCII пустой символ определен как символ с нулевым кодом (в противоположность пробелу, например, который в кодировке ASCII определен как символ с кодом 32). Команда find поддерживает операцию -print0, которая производит вывод имен файлов, разделенных пустым символом, а команда xargs имеет параметр --null, позволяющий организовать прием значений, разделенных пустым символом. Например:

find ~ -iname '\*.jpg' -print0 | xargs --null ls -l

Этот прием гарантирует правильную обработку любых имен файлов, даже содержащих пробелы.

**Возвращаемся в песочницу**

Пришло время применить find для решения некоторых практических (почти) задач. Сначала создадим песочницу с множеством файлов и каталогов:

[me@linuxbox ~]$ **mkdir -p playground/dir-{00{1..9},0{10..99},100}**

[me@linuxbox ~]$ **touch playground/dir-{00{1..9},0{10..99},100}/file-{A..Z}**

Какая мощь командной строки! Эти две строки создают каталог playground, содержащий 100 подкаталогов и 26 пустых файлов в каждом. Попробуйте-ка то же самое сделать в графическом интерфейсе!

Это волшебство мы сотворили с помощью уже знакомой команды (mkdir) механизма подстановки в командной оболочке (фигурные скобки) и новой команды touch. Объединив команду mkdir с параметром -p (который вынуждает mkdir создать родительские каталоги в указанном пути) с подстановкой фигурных скобок, мы смогли создать 100 каталогов.

Команда touch обычно используется для обновления времени последнего изменения файлов. Но если передать ей имя несуществующего файла, она создаст пустой файл.

В нашей песочнице мы создали 100 файлов с именем *file-A*. Давайте найдем их:

[me@linuxbox ~]$ **find playground -type f -name 'file-A'**

Обратите внимание, что, в отличие от ls, find возвращает результаты в несортированном порядке. Порядок определяется организацией устройства хранения. Мы можем убедиться, что действительно имеем 100 файлов с именем *file-A*:

[me@linuxbox ~]$ **find playground -type f -name 'file-A' | wc -l**

100

А теперь выполним поиск файлов по времени их последнего изменения. Этот подход можно использовать для создания резервных копий или организации файлов в хронологическом порядке. Для этого сначала создадим эталонный файл, время последнего изменения которого будет использоваться для сравнения:

[me@linuxbox ~]$ **touch playground/timestamp**

Эта команда создаст пустой файл *timestamp* и установит время его последнего изменения равным текущему времени. Мы можем убедиться в этом, использовав еще одну полезную команду, stat, которую можно рассматривать как своего рода форсированную версию ls. Команда stat выводит всю информацию о файле и его атрибутах, которой обладает система:

[me@linuxbox ~]$ **stat playground/timestamp**

  File: `playground/timestamp'

  Size: 0          Blocks: 0       IO Block: 4096 regular empty file

Device: 803h/2051d Inode: 14265061 Links: 1

Access: (0644/-rw-r--r--)  Uid: ( 1001/ me)   Gid: ( 1001/ me)

Access: 2012-10-08 15:15:39.000000000 -0400

Modify: 2012-10-08 15:15:39.000000000 -0400

Change: 2012-10-08 15:15:39.000000000 -0400

Если применить команду touch к файлу еще раз и затем исследовать его с помощью stat, мы увидим, что время последнего его изменения обновилось:

[me@linuxbox ~]$ **touch playground/timestamp**

[me@linuxbox ~]$ **stat playground/timestamp**

  File: `playground/timestamp'

  Size: 0          Blocks: 0       IO Block: 4096 regular empty file

Device: 803h/2051d Inode: 14265061 Links: 1

Access: (0644/-rw-r--r--)  Uid: ( 1001/ me)   Gid: ( 1001/ me)

Access: 2012-10-08 15:23:33.000000000 -0400

Modify: 2012-10-08 15:23:33.000000000 -0400

Change: 2012-10-08 15:23:33.000000000 -0400

Далее воспользуемся командой find, чтобы обновить время последнего изменения некоторых файлов в нашей песочнице:

[me@linuxbox ~]$ **find playground -type f -name 'file-B' -exec touch '{}' ';'**

Эта команда обновит время последнего изменения для всех файлов с именем *file-B*, имеющихся в песочнице. Теперь найдем с помощью find обновленные файлы, сравнив все файлы с эталонным файлом *timestamp*:

[me@linuxbox ~]$ **find playground -type f -newer playground/timestamp**

В результате мы получим все 100 файлов с именем *file-B*. Поскольку команда touch применялась ко всем файлм *file-B* в песочнице после обновления файла *timestamp*, они оказались «новее», чем *timestamp*, и потому были идентифицированы проверкой -newer.

В заключение вернемся к проверке плохих разрешений, выполнявшейся выше, и применим ее к каталогу *playground*:

[me@linuxbox ~]$ **find playground \( -type f -not -perm 0600 \) -or \( -type d -not -perm 0700 \)**

199

Эта команда выведет все 100 каталогов и 2600 файлов, хранящихся в *playground* (а также файл *timestamp* и сам каталог *playground*, всего 2702 элемента), потому что ни один из них не соответствует нашему определению «удовлетворительные разрешения». Вооружившись новыми знаниями об операторах и операциях, добавим в эту команду операции для применения новых разрешений к файлам и каталогам в песочнице:

[me@linuxbox ~]$ **find playground \( -type f -not -perm 0600 -exec chmod 0600**

**'{}' ';' \) -or \( -type d -not -perm 0700 -exec chmod 0700 '{}' ';' \)**

Основываясь на повседневном опыте, следует отметить, что намного проще ввести две команды — одну для каталогов и одну для файлов, чем одну большую составную команду, но знание, что можно действовать именно так, вам не помешает. Главное, что вы должны понять, — как можно использовать операторы и операции для решения практических задач.

**Параметры**

Наконец мы добрались до параметров. Параметры помогают управлять областью поиска. Они могут включаться в выражения команды find наряду с другими проверками и операциями. В табл. 17.8 перечислены наиболее часто используемые параметры.

**Таблица 17.8. Параметры команды find**

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| -depth | Требует от find обработать сначала файлы в каталогах и только потом каталоги. Этот параметр автоматически применяется с операцией -delete |
| -maxdepth число\_уровней | Устанавливает максимальное число уровней, на которое команда find может опускаться в дереве каталогов, выполняя проверки и операции |
| -mindepth число\_уровней | Устанавливает минимальное число уровней, на которое команда find должна опуститься в дереве каталогов перед выполнением проверок и операций |
| -mount | Требует от find не выполнять обход каталогов, в которые смонтированы другие файловые системы |
| -noleaf | Требует от find не оптимизировать поиск, опираясь на предположение, что поиск ведется в Unix-подобной файловой системе. Этот параметр необходимо использовать при обходе файловых систем DOS/Windows CD-ROM |