**4. Операции с файлами и каталогами**

Теперь мы готовы приступить к настоящей работе! В этой главе будут представлены следующие команды:

• cp — копирует файлы и каталоги.

• mv — перемещает/переименовывает файлы и каталоги.

• mkdir — создает каталоги.

• rm — удаляет файлы и каталоги.

• ln — создает жесткие и символические ссылки.

Эти пять команд относятся к числу наиболее часто используемых в Linux. Они применяются для управления файлами и каталогами.

Справедливости ради следует заметить, что некоторые задачи, выполняемые этими командами, гораздо проще решаются с помощью графического диспетчера файлов. В диспетчере файлов можно мышью перетаскивать файлы из одного каталога в другой, вырезать и вставлять файлы, удалять файлы и т.д. Но зачем тогда использовать эти старые программы командной строки?

Ответ прост: потому что они обладают мощностью и гибкостью. Несмотря на то что простые операции с файлами легко выполняются в диспетчере файлов с графическим интерфейсом, сложные задачи проще решать с помощью программ командной строки. Например, как скопировать файлы HTML из одного каталога в другой, причем только те, что отсутствуют в каталоге назначения или имеют более позднюю дату последнего изменения? Сделать это в диспетчере файлов очень сложно, но легко в командной строке:

cp -u \*.html destination

**Групповые символы**

Прежде чем приступать к использованию обсуждаемых команд, необходимо сначала поговорить об одной особенности командной оболочки, которая делает эти команды такими мощными. Так как имена файлов используются в командной оболочке повсеместно, она поддерживает специальные символы, помогающие быстро определять группы имен файлов. Эти специальные символы называют *групповыми символами* (wildcards). Групповые символы (также известны как *символы подстановки* (globbing)) позволяют выбирать имена файлов по шаблону. В табл. 4.1 перечислены групповые символы и их соответствия.

**Таблица 4.1. Групповые символы**

|  |  |
| --- | --- |
| **Групповой символ** | **Соответствует** |
| \* | Любая последовательность любых символов |
| ? | Любой один символ |
| [*символы*] | Любой один символ из указанного множества *символов* |
| [!*символы*] | Любой один символ, не принадлежащий указанному множеству *символов* |
| [[:*класс*:]] | Любой один символ, принадлежащий указанному *классу* |

В табл. 4.2 представлены наиболее часто используемые классы символов.

**Таблица 4.2. Наиболее часто используемые классы символов**

|  |  |
| --- | --- |
| **Класс символов** | **Соответствует** |
| [:alnum:] | Любой алфавитно-цифровой символ |
| [:alpha:] | Любой алфавитный символ |
| [:digit:] | Любой цифровой символ |
| [:lower:] | Любая буква в нижнем регистре |
| [:upper:] | Любая буква в верхнем регистре |

**диапазоны символов**

Если у вас уже есть опыт работы с другим Unix-подобным окружением или вам приходилось читать другие книги по этой теме, вы встречали форму записи диапазонов символов [A-Z] или [a-z]. Это традиционные для Unix формы записи, и они прекрасно работают в старых версиях Linux. Более того, они все еще работают в новых версиях, но будьте очень осторожны при их использовании, потому что они не всегда дают ожидаемый результат без правильной настройки. А вообще, старайтесь избегать их и использовать классы символов.

Групповые символы позволяют конструировать сложные критерии выбора имен файлов. В табл. 4.3 перечислены некоторые примеры шаблонов и их соответствия.

**Таблица 4.3. Примеры использования групповых символов**

|  |  |
| --- | --- |
| **Шаблон** | **Соответствует** |
| \* | Все имена файлов |
| g\* | Все имена файлов, начинающиеся с символа «*g*» |
| b\*.txt | Все имена файлов, начинающиеся с символа «*b*», за которым следует любое число других символов, и заканчивающиеся на «*.txt*» |
| Data??? | Все имена файлов, начинающиеся с символов «*Data*», за которыми следуют ровно три любых символа |
| [abc]\* | Все имена файлов, начинающиеся с символа «*a*», «*b*» или «*c*» |
| BACKUP.[0-9][0-9][0-9] | Все имена файлов, начинающиеся с символов «*BACKUP.*», за которыми следуют ровно три цифровых символа |
| [[:upper:]]\* | Все имена файлов, начинающиеся с буквы в верхнем регистре |
| [![:digit:]]\* | Все имена файлов, не начинающиеся с цифры |
| \*[[:lower:]123] | Все имена файлов, заканчивающиеся буквой в нижнем регистре или цифрой «*1*», «*2*» или «*3*» |

Групповые символы можно использовать с любыми командами, принимающими имена файлов в виде аргументов, но подробнее об этом мы поговорим в главе 7.

**групповые символы также действуют в графическом интерфейсе**

Групповые символы имеют особую ценность не только потому, что часто используются в командной строке, но и потому, что поддерживаются некоторыми диспетчерами с графическим интерфейсом.

• В **Nautilus** (диспетчер файлов для GNOME) можно выбирать файлы с помощью диалога **Edit** (Правка) —> **Select Pattern** (Выделить по шаблону). Просто введите шаблон для выбора файлов с групповыми символами, и в текущем каталоге будут выделены файлы, соответствующие шаблону.

• В некоторых версиях **Dolphin** и **Konqueror** (диспетчеры файлов для KDE) групповые символы можно вводить непосредственно в адресную строку. Например, если понадобится увидеть все файлы с именами, начинающимися с буквы «*u*» в нижнем регистре, в каталоге */usr/bin*, просто введите в адресной строке текст: **/usr/bin/u\***, и вы получите желаемый результат.

Большинство идей, первоначально реализованных в интерфейсе командной строки, перекочевали и в графический интерфейс. Это одно из множества обстоятельств, которые делают настольный компьютер с Linux таким мощным инструментом.

**mkdir — создание каталогов**

Команда mkdir используется для создания каталогов. Вызывается она следующим образом:

mkdir *каталог*...

**Примечание к форме записи:** В этой книге три точки в описании команды, следующие за аргументом (как в примере выше), говорят о том, что аргументов может быть несколько; то есть в данном случае команда

mkdir dir1

создаст один каталог с именем *dir1*, а команда

mkdir dir1 dir2 dir3

создаст три каталога с именами *dir1*, *dir2* и *dir3*.

**cp — копирование файлов и каталогов**

Команда cp копирует файлы и каталоги. Ее можно использовать двумя разными способами:

cp item1 item2

чтобы скопировать один файл или каталог *item1* в файл или каталог *item2*, и

cp *элемент*... *каталог*

чтобы скопировать несколько элементов (файлов или каталогов) в указанный каталог.

В табл. 4.4 и 4.5 перечислены некоторые параметры (короткие и эквивалентные им длинные), наиболее часто используемые с командой cp.

**Таблица 4.4. Параметры команды cp**

| **Параметр** | **Значение** |
| --- | --- |
| -a, --archive | Скопировать файлы и каталоги со всеми атрибутами, включая идентификаторы владельцев и права доступа. Без этого параметра копии обычно получают значения атрибутов по умолчанию, определенных для пользователя, выполняющего копирование |
| -i, --interactive | Запрашивать у пользователя подтверждение перед пере­записью существующего файла. **Если этот параметр отсут-** |
|  | **ствует, команда**cp**просто перезапишет существующие файлы** |
| -r, --recursive | Рекурсивно копировать каталоги и их содержимое. Это обязательный параметр (или параметр -a) при копировании каталогов |
| -u, --update | При копировании файлов из одного каталога в другой копировать только файлы, отсутствующие в каталоге назначения или более новые |
| -v, --verbose | Выводить информационные сообщения в процессе копиро- вания |

**Таблица 4.5. Примеры использования команды cp**

|  |  |
| --- | --- |
| **Команда** | **Результат** |
| cp file1 file2 | Скопирует *file1* в *file2*. Если *file2* существует, он будет затерт новым файлом *file1*. Если *file2* отсутствует, он будет создан |
| cp -i file1 file2 | То же, что и выше, но если файл *file2* существует, у пользователя будет запрошено подтверждение перед перезаписью файла |
| cp file1 file2 dir1 | Скопирует *file1* и *file2* в каталог *dir1*. Каталог *dir1* должен существовать |
| cp dir1/\* dir2 | С использованием группового символа. Скопирует все файлы из каталога *dir1* в каталог *dir2*. Каталог *dir2* должен существовать |
| cp -r dir1 dir2 | Скопирует каталог *dir1* (и все его содержимое) в каталог *dir2*. Если каталог *dir2* не существует, он будет создан и заполнен содержимым каталога *dir1* |

**mv — перемещение и переименование файлов**

Команда mv выполняет операции перемещения и переименования файлов в зависимости от особенностей использования. В любом случае исходный файл исчезает после операции. Команда mv используется почти так же, как команда cp:

mv item1 item2

перемещает или переименовывает файл или каталог *item1* в *item2*.

mv *элемент*... *каталог*

перемещает один или более элементов из одного каталога в другой.

Команда mv поддерживает множество тех же параметров, что и команда cp, как показано в табл. 4.6 и 4.7.

**Таблица 4.6. Параметры команды mv**

| **Параметр** | **Значение** |
| --- | --- |
| -i, --interactive | Запрашивать у пользователя подтверждение перед перезаписью существующего файла. **Если этот параметр отсутствует, ­команда**mv**просто перезапишет существующие файлы** |
| -u, --update | При перемещении файлов из одного каталога в другой перемещать только файлы, отсутствующие в каталоге назначения или более новые |
| -v, --verbose | Выводить информационные сообщения в процессе перемещения |

**Таблица 4.7. Примеры использования команды mv**

|  |  |
| --- | --- |
| **Команда** | **Результат** |
| mv file1 file2 | Переместит *file1* в *file2*. **Если***file2***существует, он будет заменен на новый файл***file1*. Если *file2* отсутствует, он будет создан. **В любом случае появится новый файл***file2* |
| mv -i file1 file2 | То же, что и выше, но если файл *file2* существует, у пользователя будет запрошено подтверждение перед перезаписью файла |
| mv file1 file2 dir1 | Переместит *file1* и *file2* в каталог *dir1*. Каталог *dir1* должен существовать |
| mv dir1 dir2 | Переместит каталог *dir1* (и все его содержимое) в каталог *dir2*. Если каталог *dir2* не существует, он будет создан и заполнен содержимым каталога *dir1*. Каталог *dir1* будет удален |

**rm — удаление файлов и каталогов**

Команда rm используется для удаления (remove) файлов и каталогов, например:

rm *элемент*...

где *элемент* — это один или несколько файлов или каталогов.

В табл. 4.8 и 4.9 перечислены некоторые параметры, наиболее часто используемые с командой rm.

**Таблица 4.8. Параметры команды rm**

| **Параметр** | **Значение** |
| --- | --- |
| -i, --interactive | Запрашивать у пользователя подтверждение перед удалением существующего файла. **Если этот параметр отсутствует, ­команда**rm**просто удалит существующие файлы** |
| -r, --recursive | Рекурсивно удалить каталоги. То есть вместе с каталогом будут удалены все его подкаталоги. Это обязательный параметр при удалении каталогов |
| -f, --force | Игнорировать отсутствующие файлы и не запрашивать подтверждения. Этот параметр отменяет действие параметра --interactive |
| -v, --verbose | Выводить информационные сообщения в процессе удаления |

**будьте осторожны с командой rm!**

Unix-подобные операционные системы, такие как Linux, не имеют команды, отменяющей удаление. Если вы что-то удалили командой rm, это исчезнет навсегда. Linux считает вас достаточно ответственным человеком, отдающим себе отчет в своих действиях.

Будьте особенно осторожны с групповыми символами. Рассмотрим следующий классический пример. Допустим, вы захотели удалить все файлы HTML в каталоге. Для этого вы вводите команду:

rm \*.html

которая сделает именно то, что вам нужно, но если вы случайно вставите пробел между \* и .html, как в следующей команде:

rm \* .html

rm удалит все файлы в каталоге и затем сообщит, что не нашла файла *.html*.

**Полезный совет:** всякий раз, используя групповые символы с командой rm (помимо внимательной проверки ввода!), проверьте сначала аргумент с групповым символом с командой ls. Это позволит увидеть, какие файлы будут удалены. Затем нажмите клавишу со стрелкой вверх, чтобы восстановить команду из истории, и замените ls на rm.

**Таблица 4.9. Примеры использования команды rm**

|  |  |
| --- | --- |
| **Команда** | **Результат** |
| rm file1 | Просто удалит файл *file1* |
| rm -i file1 | Перед удалением *file1* запросит подтверждение у пользователя |
| rm -r file1 dir1 | Удалит файл *file1* и каталог *dir1* со всем его содержимым |
| rm -rf file1 dir1 | То же, что и выше, но в отсутствие *file1* и/или *dir1* просто продолжит работу, не выводя никаких сообщений |

**ln — создание ссылок**

Команда ln применяется для создания жесткой или символической ссылки. Ее можно использовать одним из двух способов:

ln *файл* *ссылка*

создает жесткую ссылку.

ln -s *элемент* *ссылка*

создает символическую ссылку, где *элементом* может быть файл или каталог.

**Жесткие ссылки**

Жесткие ссылки — это первоначальный способ создания ссылок в Unix; символические ссылки — более позднее изобретение. По умолчанию каждый файл имеет одну жесткую ссылку, определяющую его имя. Создавая жесткую ссылку, мы создаем дополнительную запись в каталоге для файла. Жесткие ссылки имеют два важных ограничения.

• Жесткая ссылка не может указывать на файл за пределами собственной файловой системы. Это означает, что ссылка не может указывать на файл, находящийся в другом разделе диска.

• Жесткая ссылка не может указывать на каталог.

Жесткая ссылка неотличима от самого файла. В отличие от символических ссылок, при выводе списка с содержимым каталогов жесткие ссылки никак не выделяются. При удалении жесткой ссылки удаляется только сама ссылка, а файл остается на месте (то есть пространство, занимаемое файлом, не освобождается), пока не будут удалены все жесткие ссылки на файл.

Знать о существовании жестких ссылок важно, потому что они будут встречаться вам время от времени, но в современной практике предпочтение отдается символическим ссылкам, о которых рассказывается далее.

**Символические ссылки**

Символические ссылки были придуманы с целью преодолеть ограничения жестких ссылок. Когда создается символическая ссылка, в действительности создается файл особого типа, содержащий текстовый указатель на файл или каталог. В некотором отношении они действуют подобно ярлыкам в Windows, но, конечно же, появились задолго до ярлыков Windows. ;-)

Файл, на который указывает символическая ссылка, и сама символическая ссылка почти неотличимы друг от друга. Например, если попытаться что-то записать в символическую ссылку, запись будет выполнена в файл, на который она указывает. Однако при удалении символической ссылки удаляется только символическая ссылка, но не файл. Если удалить файл до того, как будет удалена символическая ссылка, ссылка останется на месте, но будет указывать в никуда. О таких ссылках говорят, что они «битые». Во многих реализациях команда ls выделяет битые ссылки цветом, например, красным, чтобы обратить на них внимание.

Идея ссылок может показаться странной и непонятной, но оставьте ее пока. Мы опробуем их на практике, и многое, возможно, для вас прояснится.

**Давайте построим песочницу**

Поскольку мы собираемся на практике опробовать некоторые операции с файлами, давайте выделим безопасный уголок для «игр» с командами управления файлами. Прежде всего нам понадобится каталог, в котором мы будем практиковаться. Создайте такой каталог в своем домашнем каталоге и назовите его *playground*.

**Создание каталогов**

Для создания каталогов используется команда mkdir. Чтобы создать каталог *play­ground*, проверьте сначала, находитесь ли вы в домашнем каталоге, и только потом создайте новый каталог:

[me@linuxbox ~]$ **cd**

[me@linuxbox ~]$ **mkdir playground**

Чтобы немножко украсить вашу песочницу, создайте внутри *playground* пару каталогов с именами *dir1* и *dir2*. Для этого смените текущий рабочий каталог на *playground* и выполните еще одну команду mkdir:

[me@linuxbox ~]$ **cd playground**

[me@linuxbox playground]$ **mkdir dir1 dir2**

Обратите внимание, что команда mkdir может принимать несколько аргументов, это позволяет создать два каталога одной командой.

**Копирование файлов**

Далее, добавим немного данных в нашу песочницу. Для этого скопируем какие-нибудь файлы. Командой cp скопируйте файл *passwd* из каталога */etc* в текущий рабочий каталог.

[me@linuxbox playground]$ **cp /etc/passwd .**

Обратите внимание на сокращение, обозначающее текущий рабочий каталог, — точку в конце команды. Если после этого выполнить команду ls, мы увидим наш файл:

[me@linuxbox playground]$ **ls -l**

итого 12

drwxrwxr-x 2 me  me 4096 2012-01-10 16:40 dir1

drwxrwxr-x 2 me  me 4096 2012-01-10 16:40 dir2

-rw-r--r-- 1 me  me 1650 2012-01-10 16:07 passwd

Теперь ради развлечения повторите операцию копирования, но на этот раз с параметром -v, чтобы посмотреть, как она работает:

[me@linuxbox playground]$ **cp -v /etc/passwd .**

`/etc/passwd' -> `./passwd'

Команда cp вновь скопировала файл, но на этот раз вывела короткое сообщение, указывающее, что операция была выполнена. Обратите внимание, что cp перезаписала первую копию без каких-либо предупреждений. Это как раз тот случай, когда cp полагает, что вы знаете, что делаете. Чтобы вывести предупреждение, включите параметр -i:

[me@linuxbox playground]$ **cp -i /etc/passwd .**

cp: переписать `./passwd'?

Если в ответ на запрос ввести y, команда перезапишет существующий файл; если ввести любой другой символ (например, n), cp оставит прежнюю копию файла нетронутой.

**Перемещение и переименование файлов**

Имя *passwd* не выглядит органичным в нашей песочнице, поэтому дадим этому файлу какое-нибудь другое имя:

[me@linuxbox playground]$ **mv passwd fun**

Теперь немножко позабавимся и переместим переименованный файл в каждый из каталогов и обратно:

[me@linuxbox playground]$ **mv fun dir1**

переместит файл в каталог *dir1*. Следующая команда

[me@linuxbox playground]$ **mv dir1/fun dir2**

переместит файл из каталога *dir1* в каталог *dir2*. Следующая команда

[me@linuxbox playground]$ **mv dir2/fun .**

вернет его в текущий рабочий каталог. Теперь посмотрим, как mv влияет на каталоги. Сначала переместите файл в каталог *dir1*:

[me@linuxbox playground]$ **mv fun dir1**

затем переместите *dir1* в *dir2* и проверьте их содержимое командой ls:

[me@linuxbox playground]$ **mv dir1 dir2**

[me@linuxbox playground]$ **ls -l dir2**

итого 4

drwxrwxr-x 2 me  me   4096 2012-01-11 06:06 dir1

[me@linuxbox playground]$ **ls -l dir2/dir1**

итого 4

-rw-r--r-- 1 me  me   1650 2012-01-10 16:33 fun

Обратите внимание: так как *dir2* уже существует, mv переместит *dir1* в *dir2*. Если бы каталога *dir2* не было, mv просто переименовала бы *dir1* в *dir2*. В заключение верните все на свои места:

[me@linuxbox playground]$ **mv dir2/dir1 .**

[me@linuxbox playground]$ **mv dir1/fun .**

**Создание жестких ссылок**

Теперь попробуем поиграть со ссылками. Сначала займемся жесткими ссылками: создайте несколько жестких ссылок для нашего файла:

[me@linuxbox playground]$ **ln fun fun-hard**

[me@linuxbox playground]$ **ln fun dir1/fun-hard**

[me@linuxbox playground]$ **ln fun dir2/fun-hard**

Теперь у нас есть четыре экземпляра файла *fun*. Посмотрим, что содержит наш каталог *playground*:

[me@linuxbox playground]$ **ls -l**

итого 16

drwxrwxr-x 2 me  me   4096 2012-01-14 16:17 dir1

drwxrwxr-x 2 me  me   4096 2012-01-14 16:17 dir2

-rw-r--r-- 4 me  me   1650 2012-01-10 16:33 fun

-rw-r--r-- 4 me  me   1650 2012-01-10 16:33 fun-hard

Прежде всего следует обратить внимание на на второе поле в записях, соответствующих файлам *fun* и *fun-hard*. Оба они содержат *4* — число жестких ссылок на файл, существующих в данный момент. Как вы помните, файл всегда имеет хотя бы одну жесткую ссылку, потому что имя файла определяется ссылкой. Но как убедиться, что *fun* и *fun-hard* — это один и тот же файл? В этом случае команда ls нам не помощник. Вы, конечно, скажете, что *fun* и *fun-hard* имеют одинаковые размеры (поле 5), но по списку файлов нельзя уверенно утверждать, что это один и тот же файл. Чтобы решить эту задачу, заглянем поглубже.

Рассуждая о жестких ссылках, полезно представлять файлы состоящими из двух частей: раздела с данными, где хранится содержимое файла, и раздела с именем, где хранится имя файла. Создавая жесткую ссылку, мы фактически создаем дополнительный раздел с именем, ссылающийся на тот же раздел с данными. Цепочку дисковых блоков система присваивает тому, что называется *индексным узлом* (inode), который затем присваивается разделу с именем. То есть каждая жесткая ссылка ссылается на определенный индексный узел с содержимым файла.

Команда ls может извлекать эту информацию. Для этого ее нужно вызвать с параметром -i:

[me@linuxbox playground]$ **ls -li**

итого 16

12353539 drwxrwxr-x 2 me  me   4096 2012-01-14 16:17 dir1

12353540 drwxrwxr-x 2 me  me   4096 2012-01-14 16:17 dir2

12353538 -rw-r--r-- 4 me  me   1650 2012-01-10 16:33 fun

12353538 -rw-r--r-- 4 me  me   1650 2012-01-10 16:33 fun-hard

В этой версии списка в первом поле отображается номер индексного узла, и, как можно видеть, оба имени, *fun* и *fun-hard*, ссылаются на индексные узлы с одним и тем же номером, а это подтверждает, что они соответствуют одному и тому же файлу.

**Создание символических ссылок**

Символические ссылки были придуманы с целью преодолеть ограничения жестких ссылок: жесткие ссылки не могут указывать на файлы, находящиеся на других физических устройствах, и не могут указывать на каталоги — только на файлы. Символическая ссылка — это файл особого типа, хранящий текстовый указатель на файл или каталог.

Создаются символические ссылки почти так же, как жесткие ссылки:

[me@linuxbox playground]$ **ln -s fun fun-sym**

[me@linuxbox playground]$ **ln -s ../fun dir1/fun-sym**

[me@linuxbox playground]$ **ln -s ../fun dir2/fun-sym**

Первый пример достаточно очевиден: мы просто добавили параметр -s, чтобы вместо жесткой ссылки создать символическую ссылку. Но два других выглядят несколько необычно. Не забывайте, что, создавая символическую ссылку, мы фактически определяем текст, описывающий местоположение целевого файла относительно символической ссылки. В этом легко убедиться, если взглянуть на вывод команды ls:

[me@linuxbox playground]$ **ls -l dir1**

итого 4

-rw-r--r-- 4 me  me   1650 2012-01-10 16:33 fun-hard

lrwxrwxrwx 1 me  me      6 2012-01-15 15:17 fun-sym -> ../fun

Запись с информацией о *fun-sym* в *dir1* сообщает, что это символическая ссылка (первый символ l в первом поле), указывающая на *../fun*, что правильно. Относительно символической ссылки *fun-sym* файл *fun* находится в каталоге уровнем выше. Обратите также внимание на размер файла символической ссылки, равный 6, — это число символов в строке ../fun, а не размер файла, на который она указывает.

При создании символических ссылок можно также указывать абсолютные пути, например:

[me@linuxbox playground]$ **ln -s /home/me/playground/fun dir1/fun-sym**

или относительные, как в более раннем примере. Но предпочтительнее использовать относительные пути, потому что это позволяет переименовывать и/или перемещать каталоги, содержащие символические ссылки, не разрушая их.

Помимо обычных файлов, символические ссылки могут указывать также на каталоги:

[me@linuxbox playground]$ **ln -s dir1 dir1-sym**

[me@linuxbox playground]$ **ls -l**

итого 16

drwxrwxr-x 2 me  me   4096 2012-01-15 15:17 dir1

lrwxrwxrwx 1 me  me      4 2012-01-16 14:45 dir1-sym -> dir1

drwxrwxr-x 2 me  me   4096 2012-01-15 15:17 dir2

-rw-r--r-- 4 me  me   1650 2012-01-10 16:33 fun

-rw-r--r-- 4 me  me   1650 2012-01-10 16:33 fun-hard

lrwxrwxrwx 1 me  me      3 2012-01-15 15:15 fun-sym -> fun

**Удаление файлов и каталогов**

Как уже говорилось ранее, удаление файлов и каталогов выполняется при помощи команды rm. Далее мы немного почистим нашу песочницу. Сначала удалите одну из жестких ссылок:

[me@linuxbox playground]$ **rm fun-hard**

[me@linuxbox playground]$ **ls -l**

итого 12

drwxrwxr-x 2 me  me   4096 2012-01-15 15:17 dir1

lrwxrwxrwx 1 me  me      4 2012-01-16 14:45 dir1-sym -> dir1

drwxrwxr-x 2 me  me   4096 2012-01-15 15:17 dir2

-rw-r--r-- 3 me  me   1650 2012-01-10 16:33 fun

lrwxrwxrwx 1 me  me      3 2012-01-15 15:15 fun-sym -> fun

Результат получился вполне ожидаемым. Файл *fun-hard* исчез, и счетчик ссылок во втором поле в записи для файла *fun* уменьшился с четырех до трех. Далее, удалите файл *fun* и ради развлечения добавьте в команду параметр -i, чтобы посмотреть, что происходит:

[me@linuxbox playground]$ **rm -i fun**

rm: удалить обычный файл  `fun'?

Введите y в ответ на запрос, и файл будет удален. Но давайте посмотрим на вывод ls. Заметили, что произошло с *fun-sym*? Поскольку теперь символическая ссылка указывает на несуществующий файл, она стала *битой*:

[me@linuxbox playground]$ **ls -l**

итого 8

drwxrwxr-x 2 me  me   4096 2012-01-15 15:17 dir1

lrwxrwxrwx 1 me  me      4 2012-01-16 14:45 dir1-sym -> dir1

drwxrwxr-x 2 me  me   4096 2012-01-15 15:17 dir2

lrwxrwxrwx 1 me  me      3 2012-01-15 15:15 fun-sym -> fun

В большинстве дистрибутивов Linux команда ls особым образом настраивается на отображение битых ссылок. В Fedora битые ссылки отображаются как мигающий красный текст. Битые ссылки не представляют никакой опасности, но вносят определенную путаницу. При попытке использовать битую ссылку вы увидите:

[me@linuxbox playground]$ **less fun-sym**

fun-sym: Нет такого файла или каталога

Давайте немного приберем за собой. Удалите символическую ссылку:

[me@linuxbox playground]$ **rm fun-sym dir1-sym**

[me@linuxbox playground]$ **ls -l**

итого 8

drwxrwxr-x 2 me  me   4096 2012-01-15 15:17 dir1

drwxrwxr-x 2 me  me   4096 2012-01-15 15:17 dir2

Главное, что следует помнить о символических ссылках: большинство операций с файлами воздействуют на целевой элемент, а не на саму ссылку. Однако команда rm является исключением из этого правила. Когда вы удаляете ссылку, удаляется сама ссылка, а не элемент, на который она указывает.

В заключение удалим каталог *playground*. Для этого вернитесь в домашний каталог и вызовите команду rm с параметром рекурсивного удаления каталогов (-r), чтобы удалить каталог *playground* и все его содержимое, включая подкаталоги:

[me@linuxbox playground]$ **cd**

[me@linuxbox ~]$ **rm -r playground**

**создание символических ссылок с помощью графического интерфейса**

Диспетчеры файлов в GNOME и KDE предоставляют простой автоматизированный способ создания символических ссылок. Если в GNOME во время перетаскивания файла мышью удерживать нажатыми клавиши **CTRL** и **SHIFT**, вместо копирования (или перемещения) файлов будет выполнена операция создания ссылки. В KDE, когда перетаскиваемый файл сбрасывается в целевой каталог, появляется небольшое меню, предлагающее выбор из трех операций: скопировать, переместить или создать ссылку.

**Заключительное замечание**

Мы узнали много нового, но чтобы информация усвоилась, требуется время. Выполняйте упражнения в песочнице раз за разом, пока не почувствуете, что понимаете их смысл. На данном этапе очень важно надежно усвоить, как работают основные команды управления файлами и групповые символы. Не бойтесь выйти за рамки предложенных упражнений — добавьте дополнительные файлы и каталоги, поэкспериментируйте с групповыми символами для определения групп файлов в разных операциях. Идея ссылок на первый взгляд может показаться малопонятной, поэтому уделите время их исследованию. Зачастую они оказываются настоящим спасательным кругом.