第五章操作文件和目录

At this point, we are ready for some real work! This chapter will introduce the following commands:

此时此刻，我们已经准备好了做些真正的工作！这一章节将会介绍以下命令：

* cp – Copy files and directories
* mv – Move/rename files and directories
* mkdir – Create directories
* rm – Remove files and directories
* ln – Create hard and symbolic links
* cp — 复制文件和目录
* mv — 移动/重命名文件和目录
* mkdir — 创建目录
* rm — 删除文件和目录
* ln — 创建硬链接和符号链接

These five commands are among the most frequently used Linux commands. They are used for manipulating both files and directories.

这五个命令属于最常使用的 Linux 命令之列。它们用来操作文件和目录。

Now, to be frank, some of the tasks performed by these commands are more easily done with a graphical file manager. With a file manager, we can drag and drop a file from one directory to another, cut and paste files, delete files, etc. So why use these old command line programs?

现在，坦诚地说，用图形文件管理器来完成一些由这些命令执行的任务会更容易些。使用文件管理器， 我们可以把文件从一个目录拖放到另一个目录，剪贴和粘贴文件，删除文件等等。那么， 为什么还使用早期的命令行程序呢？

The answer is power and flexibility. While it is easy to perform simple file manipulations with a graphical file manager, complicated tasks can be easier with the command line programs. For example, how could we copy all the HTML files from one directory to another, but only copy files that do not exist in the destination directory or are newer than the versions in the destination directory? Pretty hard with a file manager. Pretty easy with the command line:

答案是命令行程序，功能强大灵活。虽然图形文件管理器能轻松地实现简单的文件操作，但是对于 复杂的文件操作任务，则使用命令行程序比较容易完成。例如，怎样复制一个目录下的 HTML 文件 到另一个目录，但这些 HTML 文件不存在于目标目录，或者是文件版本新于目标目录里的文件？ 要完成这个任务，使用文件管理器相当难，使用命令行相当容易：

cp -u \*.html destination

**通配符**

Before we begin using our commands, we need to talk about a shell feature that makes these commands so powerful. Since the shell uses filenames so much, it provides special characters to help you rapidly specify groups of filenames. These special characters are called wildcards. Using wildcards (which is also known as globbing) allow you to select filenames based on patterns of characters. The table below lists the wildcards and what they select:

在开始使用命令之前，我们需要介绍一个使命令行如此强大的 shell 特性。因为 shell 频繁地使用 文件名，shell 提供了特殊字符来帮助你快速指定一组文件名。这些特殊字符叫做通配符。 使用通配符（也以文件名代换著称）允许你依据字符类型来选择文件名。下表列出这些通配符 以及它们所选择的对象：

|  |  |
| --- | --- |
| *Table 5-1: Wildcards* | |
| Wildcard | Meaning |
| \* | Matches any characters |
| ? | Matches any single character |
| [characters] | Matches any character that is a member of the set characters |
| [!characters] | Matches any character that is not a member of the set characters |
| [[:class:]] | Matches any character that is a member of the specified class |

|  |  |
| --- | --- |
| *表5-1: 通配符* | |
| 通配符 | 意义 |
| \* | 匹配任意多个字符（包括零个或一个） |
| ? | 匹配任意一个字符（不包括零个） |
| [characters] | 匹配任意一个属于字符集中的字符 |
| [!characters] | 匹配任意一个不是字符集中的字符 |
| [[:class:]] | 匹配任意一个属于指定字符类中的字符 |

Table 5-2 lists the most commonly used character classes:

表5-2列出了最常使用的字符类：

|  |  |
| --- | --- |
| *Table 5-2: Commonly Used Character Classes* | |
| Character Class | Meaning |
| [:alnum:] | Matches any alphanumeric character |
| [:alpha:] | Matches any alphabetic character |
| [:digit:] | Matches any numeral |
| [:lower:] | Matches any lowercase letter |
| [:upper] | Matches any uppercase letter |

|  |  |
| --- | --- |
| *表5-2: 普遍使用的字符类* | |
| 字符类 | 意义 |
| [:alnum:] | 匹配任意一个字母或数字 |
| [:alpha:] | 匹配任意一个字母 |
| [:digit:] | 匹配任意一个数字 |
| [:lower:] | 匹配任意一个小写字母 |
| [:upper] | 匹配任意一个大写字母 |

Using wildcards makes it possible to construct very sophisticated selection criteria for filenames. Here are some examples of patterns and what they match:

借助通配符，为文件名构建非常复杂的选择标准成为可能。下面是一些类型匹配的范例:

|  |  |
| --- | --- |
| *Table 5-3: Wildcard Examples* | |
| Pattern | Matches |
| \* | All files |
| g\* | All file beginning with "g" |
| b\*.txt | Any file beginning with "b" followed by any characters and ending with ".txt" |
| Data??? | Any file beginning with "Data" followed by exactly three characters |
| [abc]\* | Any file beginning with either an "a", a "b", or a "c" |
| BACKUP.[0-9][0-9][0-9] | Any file beginning with "BACKUP." followed by exactly three numerals |
| [[:upper:]]\* | Any file beginning with an uppercase letter |
| [![:digit:]]\* | Any file not beginning with a numeral |
| \*[[:lower:]123] | Any file ending with a lowercase letter or the numerals "1", "2", or "3" |

|  |  |
| --- | --- |
| *表5-3: 通配符范例* | |
| 模式 | 匹配对象 |
| \* | 所有文件 |
| g\* | 文件名以“g”开头的文件 |
| b\*.txt | 以"b"开头，中间有零个或任意多个字符，并以".txt"结尾的文件 |
| Data??? | 以“Data”开头，其后紧接着3个字符的文件 |
| [abc]\* | 文件名以"a","b",或"c"开头的文件 |
| BACKUP.[0-9][0-9][0-9] | 以"BACKUP."开头，并紧接着3个数字的文件 |
| [[:upper:]]\* | 以大写字母开头的文件 |
| [![:digit:]]\* | 不以数字开头的文件 |
| \*[[:lower:]123] | 文件名以小写字母结尾，或以 “1”，“2”，或 “3” 结尾的文件 |

Wildcards can be used with any command that accepts filenames as arguments, but we’ll talk more about that in Chapter 8.

接受文件名作为参数的任何命令，都可以使用通配符，我们会在第八章更深入的谈到这个知识点。

Character Ranges

字符范围

If you are coming from another Unix-like environment or have been reading some other books on this subject, you may have encountered the [A-Z] or the [a-z] character range notations. These are traditional Unix notations and worked in older versions of Linux as well. They can still work, but you have to be very careful with them because they will not produce the expected results unless properly configured. For now, you should avoid using them and use character classes instead.

如果你用过别的类 Unix 系统的操作环境，或者是读过这方面的书籍，你可能遇到过[A-Z]或 [a-z]形式的字符范围表示法。这些都是传统的 Unix 表示法，并且在早期的 Linux 版本中仍有效。 虽然它们仍然起作用，但是你必须小心地使用它们，因为它们不会产生你期望的输出结果，除非 你合理地配置它们。从现在开始，你应该避免使用它们，并且用字符类来代替它们。

Wildcards Work In The GUI Too

通配符在 GUI 中也有效

Wildcards are especially valuable not only because they are used so frequently on the command line, but are also supported by some graphical file managers.

通配符非常重要，不仅因为它们经常用在命令行中，而且一些图形文件管理器也支持它们。

* In Nautilus (the file manager for GNOME), you can select files using the Edit/Select Pattern menu item. Just enter a file selection pattern with wildcards and the files in the currently viewed directory will be highlighted for selection.
* In Dolphin and Konqueror (the file managers for KDE), you can enter wildcards directly on the location bar. For example, if you want to see all the files starting with a lowercase “u” in the /usr/bin directory, type “/usr/bin/u\*” into the location bar and it will display the result.
* 在 Nautilus (GNOME 文件管理器）中，可以通过 Edit/Select 模式菜单项来选择文件。 输入一个用通配符表示的文件选择模式后，那么当前所浏览的目录中，所匹配的文件名就会高亮显示。
* 在 Dolphin 和 Konqueror（KDE 文件管理器）中，可以在地址栏中直接输入通配符。例如， 如果你想查看目录 /usr/bin 中，所有以小写字母 ‘u’ 开头的文件， 在地址栏中敲入 ‘/usr/bin/u\*‘，则 文件管理器会显示匹配的结果。

Many ideas originally found in the command line interface make their way into the graphical interface, too. It is one of the many things that make the Linux desktop so powerful.

最初源于命令行界面中的想法，在图形界面中也适用。这就是使 Linux 桌面系统 如此强大的众多原因中的一个

**mkdir - 创建目录**

The mkdir command is used to create directories. It works like this:

mkdir 命令是用来创建目录的。它这样工作：

mkdir directory...

**A note on notation:** When three periods follow an argument in the description of a command (as above), it means that the argument can be repeated, thus:

**注意表示法:** 在描述一个命令时（如上所示），当有三个圆点跟在一个命令的参数后面， 这意味着那个参数可以重复，就像这样：

mkdir dir1

would create a single directory named “dir1”, while

会创建一个名为”dir1”的目录，而

mkdir dir1 dir2 dir3

would create three directokries named “dir1”, “dir2”, “dir3”.

会创建三个目录，名为 dir1, dir2, dir3。

**cp - 复制文件和目录**

The cp command copies files or directories. It can be used two different ways:

cp 命令，复制文件或者目录。它有两种使用方法：

cp item1 item2

to copy the single file or directory “item1” to file or directory “item2” and:

复制单个文件或目录”item1”到文件或目录”item2”，和：

cp item... directory

to copy multiple items (either files or directories) into a directory.

复制多个项目（文件或目录）到一个目录下。

**有用的选项和实例**

Here are some of the commonly used options (the short option and the equivalent long option) for cp:

这里列举了 cp 命令一些有用的选项（短选项和等效的长选项）：

|  |  |
| --- | --- |
| *Table 5-4: cp Options* | |
| Option | Meaning |
| -a, --archive | Copy the files and directories and all of their attributes, including ownerships and permissions. Normally, copies take on the default attributes of the user performing the copy |
| -i, --interactive | Before overwriting an existing file, prompt the user for confirmation. If this option is not specified, cp will silently overwrite files. |
| -r, --recursive | Recursively copy directories and their contents. This option (or the -a option) is required when copying directories. |
| -u, --update | When copying files from one directory to another, only copy files that either don't exist, or are newer than the existing corresponding files, in the destination directory. |
| -v, --verbose | Display informative messages as the copy is performed. |

|  |  |
| --- | --- |
| *表5-4: cp 选项* | |
| 选项 | 意义 |
| -a, --archive | 复制文件和目录，以及它们的属性，包括所有权和权限。 通常，复本具有用户所操作文件的默认属性。 |
| -i, --interactive | 在重写已存在文件之前，提示用户确认。如果这个选项不指定， cp 命令会默认重写文件。 |
| -r, --recursive | 递归地复制目录及目录中的内容。当复制目录时， 需要这个选项（或者-a 选项）。 |
| -u, --update | 当把文件从一个目录复制到另一个目录时，仅复制 目标目录中不存在的文件，或者是文件内容新于目标目录中已经存在的文件。 |
| -v, --verbose | 显示翔实的命令操作信息 |

|  |  |
| --- | --- |
| *Table 5-5: cp Examples* | |
| Command | Results |
| cp file1 file2 | Copy file1 to file2. If file2 exists, it is overwritten with the contents of file1. If file2 does not exist, it is created. |
| cp -i file1 file2 | Same as above, except that if file2 exists, the user is prompted before it is overwritten. |
| cp file1 file2 dir1 | Copy file1 and file2 into directory dir1. dir1 must already exist. |
| cp dir1/\* dir2 | Using a wildcard, all the files in dir1 are copied into dir2. dir2 must already exist. |
| cp -r dir1 dir2 | Copy the contents of directory dir1 to directory dir2. If directory dir2 does not exist, it is created and, after the copy, will contain the same contents as directory dir1. If directory dir2 does exist, then directory dir1 (and its contents) will be copied into dir2. |

|  |  |
| --- | --- |
| *表5-5: cp 实例* | |
| 命令 | 运行结果 |
| cp file1 file2 | 复制文件 file1 内容到文件 file2。如果 file2 已经存在， file2 的内容会被 file1 的内容重写。如果 file2 不存在，则会创建 file2。 |
| cp -i file1 file2 | 这条命令和上面的命令一样，除了如果文件 file2 存在的话，在文件 file2 被重写之前， 会提示用户确认信息。 |
| cp file1 file2 dir1 | 复制文件 file1 和文件 file2 到目录 dir1。目录 dir1 必须存在。 |
| cp dir1/\* dir2 | 使用一个通配符，在目录 dir1 中的所有文件都被复制到目录 dir2 中。 dir2 必须已经存在。 |
| cp -r dir1 dir2 | 复制目录 dir1 中的内容到目录 dir2。如果目录 dir2 不存在， 创建目录 dir2，操作完成后，目录 dir2 中的内容和 dir1 中的一样。 如果目录 dir2 存在，则目录 dir1 (和目录中的内容)将会被复制到 dir2 中。 |

**mv - 移动和重命名文件**

The mv command performs both file moving and file renaming, depending on how it is used. In either case, the original filename no longer exists after the operation. mv is used in much the same way as cp:

mv 命令可以执行文件移动和文件命名任务，这依赖于你怎样使用它。任何一种 情况下，完成操作之后，原来的文件名不再存在。mv 使用方法与 cp 很相像：

mv item1 item2

to move or rename file or directory “item1” to “item2” or:

把文件或目录 “item1” 移动或重命名为 “item2”, 或者：

mv item... directory

to move one or more items from one directory to another.

把一个或多个条目从一个目录移动到另一个目录中。

**有用的选项和实例**

mv shares many of the same options as cp:

mv 与 cp 共享了很多一样的选项：

|  |  |
| --- | --- |
| *Table 5-6: mv options* | |
| Option | Meaning |
| -i --interactive | Before overwriting an existing file, prompt the user for confirmation. **If this option is not specified, mv command will silently overwrite files** |
| -u --update | When moving files from one directory to another, only move files that either don't exist, or are newer than the existing corresponding files in the destination directory. |
| -v --verbose | Display informative messages as the move is performed. |

|  |  |
| --- | --- |
| *表5-6: mv 选项* | |
| 选项 | 意义 |
| -i --interactive | 在重写一个已经存在的文件之前，提示用户确认信息。 **如果不指定这个选项，mv 命令会默认重写文件内容。** |
| -u --update | 当把文件从一个目录移动另一个目录时，只是移动不存在的文件， 或者文件内容新于目标目录相对应文件的内容。 |
| -v --verbose | 当操作 mv 命令时，显示翔实的操作信息。 |

|  |  |
| --- | --- |
| *Table 5-7: mv Examples* | |
| mv file1 file2 | Move file1 to file2. **If file2 exists, it is overwritten with the contents of files.**If file2 does not exist, it is created. **In either case, file1 ceases to exist.** |
| mv -i file1 file2 | Same as above, except that if file2 exists, the user is prompted before it is overwritten. |
| mv file1 file2 dir1 | Move file1 and file2 into dirctory dir1. dir1 must already exist. |
| mv dir1 dir2 | if directory dir2 does not exist, create directory dir2 and move the contents of directory dir1 into dir2 and delete directory dir1. if directory dir2 does exist, move directory dir1 (and its contents) into directory dir2. |

|  |  |
| --- | --- |
| *表5-7: mv 实例* | |
| mv file1 file2 | 移动 file1 到 file2。**如果 file2 存在，它的内容会被 file1 的内容重写。**如果 file2 不存在，则创建 file2。**每种情况下，file1 不再存在。** |
| mv -i file1 file2 | 除了如果 file2 存在的话，在 file2 被重写之前，用户会得到 提示信息外，这个和上面的选项一样。 |
| mv file1 file2 dir1 | 移动 file1 和 file2 到目录 dir1 中。dir1 必须已经存在。 |
| mv dir1 dir2 | 如果目录 dir2 不存在，创建目录 dir2，并且移动目录 dir1 的内容到 目录 dir2 中，同时删除目录 dir1。如果目录 dir2 存在，移动目录 dir1（及它的内容）到目录 dir2。 |

**rm - 删除文件和目录**

The rm command is used to remove(delete)files and directories:

rm 命令用来移除（删除）文件和目录：

rm item...

where “item” is one or more files or directories.

“item”代表一个或多个文件或目录。

**有用的选项和实例**

Here are some of the common options for rm:

下表是一些普遍使用的 rm 选项：

|  |  |
| --- | --- |
| *Table 5-8: rm Options* | |
| Option | Meaning |
| -i, --interactive | Before deleting an existing file, prompt the user for confirmation. **If this option is not specified, rm will silently delete files.** |
| -r, --recursive | Recursively delete directories. This means that if a directory being deleted has subdirectories, delete them too. To delete a directory, this option must be specified. |
| -f, --force | Ignore nonexistent files and do not prompt. This overrides the --interactive option. |
| -v, --verbose | Display informative messages as the deletion is performed. |

|  |  |
| --- | --- |
| *表5-8: rm 选项* | |
| 选项 | 意义 |
| -i, --interactive | 在删除已存在的文件前，提示用户确认信息。 **如果不指定这个选项，rm 会默默地删除文件** |
| -r, --recursive | 递归地删除文件，这意味着，如果要删除一个目录，而此目录 又包含子目录，那么子目录也会被删除。要删除一个目录，必须指定这个选项。 |
| -f, --force | 忽视不存在的文件，不显示提示信息。这选项颠覆了“--interactive”选项。 |
| -v, --verbose | 在执行 rm 命令时，显示翔实的操作信息。 |

|  |  |
| --- | --- |
| *Table 5-9: rm Examples* | |
| Command | Results |
| rm file1 | Delete file1 silently |
| rm -i file1 | Same as above, except that the user is prompted for confirmation before the deletion is performed. |
| rm -r file1 dir1 | Delete file1 and dir1 and its contents. |
| rm -rf file1 dir1 | Same as above, except that if either file1 or dir1 do not exist, rm will continue silently. |

|  |  |
| --- | --- |
| *表5-9: rm 实例* | |
| 命令 | 运行结果 |
| rm file1 | 默默地删除文件 |
| rm -i file1 | 除了在删除文件之前，提示用户确认信息之外，和上面的命令作用一样。 |
| rm -r file1 dir1 | 删除文件 file1, 目录 dir1，及 dir1 中的内容。 |
| rm -rf file1 dir1 | 同上，除了如果文件 file1，或目录 dir1 不存在的话，rm 仍会继续执行。 |

Be Careful With rm!

小心 rm!

Unix-like operating systems such as Linux do not have an undelete command. Once you delete something with rm, it’s gone. Linux assumes you’re smart and you know what you’re doing.

类 Unix 的操作系统，比如说 Linux，没有复原命令。一旦你用 rm 删除了一些东西， 它就消失了。Linux 假定你很聪明，你知道你在做什么。

Be particularly careful with wildcards. Consider this classic example. Let’s say you want to delete just the HTML files in a directory. To do this, you type:

尤其要小心通配符。思考一下这个经典的例子。假如说，你只想删除一个目录中的 HTML 文件。输入：

**rm \*.html**

which is correct, but if you accidentally place a space between the “\*” and the “.html” like so:

这是正确的，如果你不小心在 “\*” 和 “.html” 之间多输入了一个空格，就像这样：

**rm \* .html**

the rm command will delete all the files in the directory and then complain that there is no file called “.html”.

这个 rm 命令会删除目录中的所有文件，还会抱怨没有文件叫做 “.html”。

**Here is a useful tip.** Whenever you use wildcards with rm (besides carefully checking your typing!), test the wildcard first with ls. This will let you see the files that will be deleted. Then press the up arrow key to recall the command and replace the ls with rm.

**小贴士。** 无论什么时候，rm 命令用到通配符（除了仔细检查输入的内容外！）， 用 ls 命令来测试通配符。这会让你看到要删除的文件列表。然后按下上箭头按键，重新调用 刚刚执行的命令，用 rm 替换 ls。

**ln — 创建链接**

The ln command is used to create either hard or symbolic links. It is used in one of two ways:

ln 命令既可创建硬链接，也可以创建符号链接。可以用其中一种方法来使用它：

ln file link

to create a hard link, and:

创建硬链接，和：

ln -s item link

to create a symbolic link where “item” is either a file or a directory.

创建符号链接，”item” 可以是一个文件或是一个目录。

**硬链接**

Hard links are the original Unix way of creating links, compared to symbolic links, which are more modern. By default, every file has a single hard link that gives the file its name. When we create a hard link, we create an additional directory entry for a file. Hard links have two important limitations:

硬链接和符号链接比起来，硬链接是最初 Unix 创建链接的方式，而符号链接更加现代。 在默认情况下，每个文件有一个硬链接，这个硬链接给文件起名字。当我们创建一个 硬链接以后，就为文件创建了一个额外的目录条目。硬链接有两个重要局限性：

1. A hard link cannot reference a file outside its own file system. This means a link may not reference a file that is not on the same disk partition as the link itself.
2. A hard link may not reference a directory.
3. 一个硬链接不能关联它所在文件系统之外的文件。这是说一个链接不能关联 与链接本身不在同一个磁盘分区上的文件。
4. 一个硬链接不能关联一个目录。

A hard link is indistinguishable from the file itself. Unlike a symbolic link, when you list a directory containing a hard link you will see no special indication of the link. When a hard link is deleted, the link is removed but the contents of the file itself continue to exist (that is, its space is not deallocated) until all links to the file are deleted. It is important to be aware of hard links because you might encounter them from time to time, but modern practice prefers symbolic links, which we will cover next.

一个硬链接和文件本身没有什么区别。不像符号链接，当你列出一个包含硬链接的目录 内容时，你会看到没有特殊的链接指示说明。当一个硬链接被删除时，这个链接 被删除，但是文件本身的内容仍然存在（这是说，它所占的磁盘空间不会被重新分配）， 直到所有关联这个文件的链接都删除掉。知道硬链接很重要，因为你可能有时 会遇到它们，但现在实际中更喜欢使用符号链接，下一步我们会讨论符号链接。

**符号链接**

Symbolic links were created to overcome the limitations of hard links. Symbolic links work by creating a special type of file that contains a text pointer to the referenced file or directory. In this regard, they operate in much the same way as a Windows shortcut though of course, they predate the Windows feature by many years ;-)

创建符号链接是为了克服硬链接的局限性。符号链接生效，是通过创建一个 特殊类型的文件，这个文件包含一个关联文件或目录的文本指针。在这一方面， 它们和 Windows 的快捷方式差不多，当然，符号链接早于 Windows 的快捷方式 很多年;-)

A file pointed to by a symbolic link, and the symbolic link itself are largely indistinguishable from one another. For example, if you write some something to the symbolic link, the referenced file is also written to. However when you delete a symbolic link, only the link is deleted, not the file itself. If the file is deleted before the symbolic link, the link will continue to exist, but will point to nothing. In this case, the link is said to be broken. In many implementations, the ls command will display broken links in a distinguishing color, such as red, to reveal their presence.

一个符号链接指向一个文件，而且这个符号链接本身与其它的符号链接几乎没有区别。 例如，如果你往一个符号链接里面写入东西，那么相关联的文件也被写入。然而， 当你删除一个符号链接时，只有这个链接被删除，而不是文件自身。如果先于符号链接 删除文件，这个链接仍然存在，但是不指向任何东西。在这种情况下，这个链接被称为 坏链接。在许多实现中，ls 命令会以不同的颜色展示坏链接，比如说红色，来显示它们 的存在。

The concept of links can seem very confusing, but hang in there. We’re going to try all this stuff and it will, hopefully, become clear.

关于链接的概念，看起来很迷惑，但不要胆怯。我们将要试着练习 这些命令，希望，它变得清晰起来。

**创建游戏场（实战演习）**

Since we are going to do some real file manipulation, let’s build a safe place to “play” with our file manipulation commands. First we need a directory to work in. We’ll create one in our home directory and call it “playground.”

下面我们将要做些真正的文件操作，让我们先建立一个安全地带， 来玩一下文件操作命令。首先，我们需要一个工作目录。在我们的 家目录下创建一个叫做“playground”的目录。

**创建目录**

The mkdir command is used to create a directory. To create our playground directory we will first make sure we are in our home directory and will then create the new directory:

mkdir 命令被用来创建目录。首先确定我们在我们的家目录下，来创建 playground 目录， 然后创建这个新目录：

[me@linuxbox ~]$ cd

[me@linuxbox ~]$ mkdir playground

To make our playground a little more interesting, let’s create a couple of directories inside it called “dir1” and “dir2”. To do this, we will change our current working directory to playground and execute another mkdir:

为了让我们的游戏场更加有趣，在 playground 目录下创建一对目录 ，分别叫做 “dir1” 和 “dir2”。更改我们的当前工作目录到 playground，然后 执行 mkdir 命令：

[me@linuxbox ~]$ cd playground

[me@linuxbox playground]$ mkdir dir1 dir2

Notice that the mkdir command will accept multiple arguments allowing us to create both directories with a single command.

注意到 mkdir 命令可以接受多个参数，它允许我们用一个命令来创建这两个目录。

###　复制文件

Next, let’s get some data into our playground. We’ll do this by copying a file. Using the cp command, we’ll copy the passwd file from the /etc directory to the current working directory:

下一步，让我们得到一些数据到我们的游戏场中。通过复制一个文件来实现目的。 使用 cp 命令，我们从 /etc 目录复制 passwd 文件到当前工作目录下：

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

Notice how we used the shorthand for the current working directory, the single trailing period. So now if we perform an ls, we will see our file:

注意：我们怎样使用当前工作目录的快捷方式，命令末尾的单个圆点。如果我们执行 ls 命令， 可以看到我们的文件：

[me@linuxbox playground]$ ls -l

total 12

drwxrwxr-x 2 me me 4096 2008-01-10 16:40 dir1

drwxrwxr-x 2 me me 4096 2008-01-10 16:40 dir2

-rw-r--r-- 1 me me 1650 2008-01-10 16:07 passwd

Now, just for fun, let’s repeat the copy using the “-v” option (verbose) to see what it does:

现在，仅仅是为了高兴，重复操作复制命令，使用”-v”选项（唠叨），看一个它的作用：

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

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

The cp command performed the copy again, but this time displayed a concise message indicating what operation it was performing. Notice that cp overwrote the first copy without any warning. Again this is a case of cp assuming that you know what you’re are doing. To get a warning, we’ll include the “-i” (interactive) option:

cp 命令再一次执行了复制操作，但是这次显示了一条简洁的信息，指明它 进行了什么操作。注意，cp 没有警告，就重写了第一次复制的文件。这是一个案例， cp 假定你知道你的所作所为。为了得到警示信息，在命令中包含”-i”选项：

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

cp: overwrite `./passwd'?

Responding to the prompt by entering a “y” will cause the file to be overwritten, any other character (for example, “n”) will cause cp to leave the file alone.

响应命令提示信息，输入”y”，文件就会被重写，其它的字符（例如，”n”）会导致 cp 命令不理会文件。

**移动和重命名文件**

Now, the name “passwd” doesn’t seem very playful and this is a playground, so let’s change it to something else:

现在，”passwd” 这个名字，看起来不怎么有趣，这是个游戏场，所以我们给它改个名字：

[me@linuxbox playground]$ mv passwd fun

Let’s pass the fun around a little by moving our renamed file to each of the directories and back again:

让我们来传送 fun 文件，通过移动重命名的文件到各个子目录， 然后再把它移回到当前目录：

[me@linuxbox playground]$ mv fun dir1

to move it first to directory dir1, then:

首先，把 fun 文件移动目录 dir1 中，然后：

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

to move it from dir1 to dir2, then:

再把 fun 文件从 dir1 移到目录 dir2, 然后：

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

to finally bringing it back to the current working directory. Next, let’s see the effect of mv on directories. First we will move our data file into dir1 again:

最后，再把 fun 文件带回到当前工作目录。下一步，来看看移动目录的效果。 首先，我们先移动我们的数据文件到 dir1 目录：

[me@linuxbox playground]$ mv fun dir1

then move dir1 into dir2 and confirm it with ls:

然后移动 dir1到 dir2目录，用 ls 来确认执行结果:

[me@linuxbox playground]$ mv dir1 dir2

[me@linuxbox playground]$ ls -l dir2

total 4

drwxrwxr-x 2 me me 4096 2008-01-11 06:06 dir1

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

total 4

-rw-r--r-- 1 me me 1650 2008-01-10 16:33 fun

Note that since dir2 already existed, mv moved dir1 into dir2. If dir2 had not existed, mv would have renamed dir1 to dir2. Lastly, let’s put everything back:

注意：因为目录 dir2 已经存在，mv 命令移动 dir1 到 dir2 目录。如果 dir2 不存在， mv 会重新命名 dir1 为 dir2。最后，把所有的东西放回原处。

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

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

**创建硬链接**

Now we’ll try some links. First the hard links. We’ll create some links to our data file like so:

现在，我们试着创建链接。首先是硬链接。我们创建一些关联我们 数据文件的链接：

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

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

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

So now we have four instances of the file “fun”. Let’s take a look our playground directory:

所以现在，我们有四个文件”fun”的实例。看一下目录 playground 中的内容：

[me@linuxbox playground]$ ls -l

total 16

drwxrwxr-x 2 me me 4096 2008-01-14 16:17 dir1

drwxrwxr-x 2 me me 4096 2008-01-14 16:17 dir2

-rw-r--r-- 4 me me 1650 2008-01-10 16:33 fun

-rw-r--r-- 4 me me 1650 2008-01-10 16:33 fun-hard

One thing you notice is that the second field in the listing for fun and fun-hard both contain a “4” which is the number of hard links that now exist for the file. You’ll remember that a file will always have at least one because the file’s name is created by a link. So, how do we know that fun and fun-hard are, in fact, the same file? In this case, ls is not very helpful. While we can see that fun and fun-hard are both the same size (field 5), our listing provides no way to be sure. To solve this problem, we’re going to have to dig a little deeper.

注意到一件事，列表中，文件 fun 和 fun-hard 的第二个字段是”4”，这个数字 是文件”fun”的硬链接数目。你要记得一个文件至少有一个硬链接，因为文件 名就是由链接创建的。所以，我们怎样知道实际上 fun 和 fun-hard 是一样的文件呢？ 在这个例子里，ls 不是很有用。虽然我们能够看到 fun 和 fun-hard 文件大小一样 （第五字段），但我们的列表没有提供可靠的信息来确定（这两个文件一样）。 为了解决这个问题，我们更深入的研究一下。

When thinking about hard links, it is helpful to imagine that files are made up of two parts: the data part containing the file’s contents and the name part which holds the file’s name. When we create hard links, we are actually creating additional name parts that all refer to the same data part. The system assigns a chain of disk blocks to what is called an inode, which is then associated with the name part. Each hard link therefore refers to a specific inode containing the file’s contents.

当考虑到硬链接的时候，我们可以假设文件由两部分组成：包含文件内容的数据部分和持有文件名的名字部分 ，这将有助于我们理解这个概念。当我们创建文件硬链接的时候，实际上是为文件创建了额外的名字部分， 并且这些名字都关系到相同的数据部分。这时系统会分配一连串的磁盘给所谓的索引节点，然后索引节点与文 件名字部分相关联。因此每一个硬链接都关系到一个具体的包含文件内容的索引节点。

The ls command has a way to reveal this information. It is invoked with the “-i” option:

ls 命令有一种方法，来展示（文件索引节点）的信息。在命令中加上”-i”选项：

[me@linuxbox playground]$ ls -li

total 16

12353539 drwxrwxr-x 2 me me 4096 2008-01-14 16:17 dir1

12353540 drwxrwxr-x 2 me me 4096 2008-01-14 16:17 dir2

12353538 -rw-r--r-- 4 me me 1650 2008-01-10 16:33 fun

12353538 -rw-r--r-- 4 me me 1650 2008-01-10 16:33 fun-hard

In this version of the listing, the first field is the inode number and, as we can see, both fun and fun-hard share the same inode number, which confirms they are the same file.

在这个版本的列表中，第一字段表示文件索引节点号，正如我们所见到的， fun 和 fun-hard 共享一样的索引节点号，这就证实这两个文件是一样的文件。

**创建符号链接**

Symbolic links were created to overcome the two disadvantages of hard links: hard links cannot span physical devices and hard links cannot reference directories, only files. Symbolic links are a special type of file that contains a text pointer to the target file or directory.

建立符号链接的目的是为了克服硬链接的两个缺点：硬链接不能跨越物理设备， 硬链接不能关联目录，只能是文件。符号链接是文件的特殊类型，它包含一个指向 目标文件或目录的文本指针。

Creating symbolic links is similar to creating hard links:

符号链接的建立过程相似于创建硬链接：

[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

The first example is pretty straightforward, we simply add the “-s” option to create a symbolic link rather than a hard link. But what about the next two? Remember, when we create a symbolic link, we are creating a text description of where the target file is relative to the symbolic link. It’s easier to see if we look at the ls output:

第一个实例相当直接，在 ln 命令中，简单地加上”-s”选项就可以创建一个符号链接， 而不是一个硬链接。下面两个例子又是怎样呢？ 记住，当我们创建一个符号链接 的时候，会建立一个目标文件在哪里和符号链接有关联的文本描述。如果我们看看 ls 命令的输出结果，比较容易理解。

[me@linuxbox playground]$ ls -l dir1

total 4

-rw-r--r-- 4 me me 1650 2008-01-10 16:33 fun-hard

lrwxrwxrwx 1 me me 6 2008-01-15 15:17 fun-sym -> ../fun

The listing for fun-sym in dir1 shows that is it a symbolic link by the leading “l” in the first field and that it points to “../fun”, which is correct. Relative to the location of fun-sym, fun is in the directory above it. Notice too, that the length of the symbolic link file is 6, the number of characters in the string “../fun” rather than the length of the file to which it is pointing.

目录 dir1 中，fun-sym 的列表说明了它是一个符号链接，通过在第一字段中的首字符”l” 可知，并且它还指向”../fun”，也是正确的。相对于 fun-sym 的存储位置，fun 在它的 上一个目录。同时注意，符号链接文件的长度是6，这是字符串”../fun”所包含的字符数， 而不是符号链接所指向的文件长度。

When creating symbolic links, you can either use absolute pathnames:

当建立符号链接时，你即可以使用绝对路径名：

ln -s /home/me/playground/fun dir1/fun-sym

or relative pathnames, as we did in our earlier example. Using relative pathnames is more desirable because it allows a directory containing symbolic links to be renamed and/or moved without breaking the links.

也可用相对路径名，正如前面例题所展示的。使用相对路径名更令人满意， 因为它允许一个包含符号链接的目录重命名或移动，而不会破坏链接。

In addition to regular files, symbolic links can also reference directories:

除了普通文件，符号链接也能关联目录：

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

[me@linuxbox playground]$ ls -l

total 16

...省略

**移动文件和目录**

As we covered earlier, the rm command is used to delete files and directories. We are going to use it to clean up our playground a little bit. First, let’s delete one of our hard links:

正如我们之前讨论的，rm 命令被用来删除文件和目录。我们将要使用它 来清理一下我们的游戏场。首先，删除一个硬链接：

[me@linuxbox playground]$ rm fun-hard

[me@linuxbox playground]$ ls -l

total 12

...省略

That worked as expected. The file fun-hard is gone and the link count shown for fun is reduced from four to three, as indicated in the second field of the directory listing. Next, we’ll delete the file fun, and just for enjoyment, we’ll include the “-i” option to show what that does:

结果不出所料。文件 fun-hard 消失了，文件 fun 的链接数从4减到3，正如 目录列表第二字段所示。下一步，我们会删除文件 fun，仅为了娱乐，我们会包含”-i” 选项，看一个它的作用：

[me@linuxbox playground]$ rm -i fun

rm: remove regular file `fun'?

Enter “y” at the prompt and the file is deleted. But let’s look at the output of ls now. Noticed what happened to fun-sym? Since it’s a symbolic link pointing to a now- nonexistent file, the link is broken:

在提示符下输入”y”，删除文件。让我们看一下 ls 的输出结果。注意，fun-sym 发生了 什么事? 因为它是一个符号链接，指向已经不存在的文件，链接已经坏了：

[me@linuxbox playground]$ ls -l

total 8

drwxrwxr-x 2 me me 4096 2008-01-15 15:17 dir1

lrwxrwxrwx 1 me me 4 2008-01-16 14:45 dir1-sym -> dir1

drwxrwxr-x 2 me me 4096 2008-01-15 15:17 dir2

lrwxrwxrwx 1 me me 3 2008-01-15 15:15 fun-sym -> fun

Most Linux distributions configure ls to display broken links. On a Fedora box, broken links are displayed in blinking red text! The presence of a broken link is not, in and of itself dangerous but it is rather messy. If we try to use a broken link we will see this:

大多数 Linux 的发行版本配置 ls 显示损坏的链接。在 Fedora 系统中，坏的链接以闪烁的 红色文本显示！损坏链接的出现，并不危险，但是相当混乱。如果我们试着使用 损坏的链接，会看到以下情况：

[me@linuxbox playground]$ less fun-sym

fun-sym: No such file or directory

Let’s clean up a little. We’ll delete the symbolic links:

稍微清理一下现场。删除符号链接：

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

[me@linuxbox playground]$ ls -l

total 8

drwxrwxr-x 2 me me 4096 2008-01-15 15:17 dir1

drwxrwxr-x 2 me me 4096 2008-01-15 15:17 dir2

One thing to remember about symbolic links is that most file operations are carried out on the link’s target, not the link itself. rm is an exception. When you delete a link, it is the link that is deleted, not the target.

对于符号链接，有一点值得记住，执行的大多数文件操作是针对链接的对象，而不是链接本身。 而 rm 命令是个特例。当你删除链接的时候，删除链接本身，而不是链接的对象。

Finally, we will remove our playground. To do this, we will return to our home directory and use rm with the recursive option (-r) to delete playground and all of its contents, including its subdirectories:

最后，我们将删除我们的游戏场。为了完成这个工作，我们将返回到 我们的家目录，然后用 rm 命令加上选项(-r)，来删除目录 playground， 和目录下的所有内容，包括子目录：

[me@linuxbox playground]$ cd

[me@linuxbox ~]$ rm -r playground

Creating Symlinks With The GUI

用 GUI 来创建符号链接

The file managers in both GNOME and KDE provide an easy and automatic method of creating symbolic links. With GNOME, holding the Ctrl+Shift keys while dragging a file will create a link rather than copying (or moving) the file. In KDE, a small menu appears whenever a file is dropped, offering a choice of copying, moving, or linking the file.

文件管理器 GNOME 和 KDE 都提供了一个简单而且自动化的方法来创建符号链接。 在 GNOME 里面，当拖动文件时，同时按下 Ctrl+Shift 按键会创建一个链接，而不是 复制（或移动）文件。在 KDE 中，无论什么时候放下一个文件，会弹出一个小菜单， 这个菜单会提供复制，移动，或创建链接文件选项。

**总结**

We’ve covered a lot of ground here and it will take a while to fully sink in. Perform the playground exercise over and over until it makes sense. It is important to get a good understanding of basic file manipulation commands and wildcards. Feel free to expand on the playground exercise by adding more files and directories, using wildcards to specify files for various operations. The concept of links is a little confusing at first, but take the time to learn how they work. They can be a real lifesaver.

在这一章中，我们已经研究了许多基础知识。我们得花费一些时间来全面的理解。 反复练习 playground 例题，直到你觉得它有意义。能够良好的理解基本文件操作 命令和通配符，非常重要。空闲时，通过添加文件和目录来拓展 playground 练习， 使用通配符来为各种各样的操作命令指定文件。关于链接的概念，在刚开始接触 时会觉得有点迷惑，花些时间来学习它们是怎样工作的。它们能成为真正的救星。