1 – What Is the Shell?

When we speak of the command line, we are really referring to the *shell*. The shell is a program that takes keyboard commands and passes them to the operating system to carry out. Almost all Linux distributions supply a shell program from the GNU Project called `bash`. The name “bash” is an acronym for “Bourne Again SHell”, a reference to the fact `bash` is an enhanced replacement for `sh`, the original Unix shell program written by Steve Bourne.

Terminal Emulators

When using a graphical user interface (GUI), we need another program called a *terminal emulator* to interact with the shell. If we look through our desktop menus, we will probably find one. KDE uses `konsole` and GNOME uses `gnome-terminal`, though it's likely called simply “terminal” on our menu. A number of other terminal emulators are available for Linux, but they all basically do the same thing; give us access to the shell. You will probably develop a preference for one or another terminal emulator based on the number of bells and whistles it has.

Making Your First Keystrokes

So let's get started. Launch the terminal emulator! Once it comes up, we should see something like this:

```
[me@linuxbox ~]$
```

This is called a *shell prompt* and it will appear whenever the shell is ready to accept input. While it may vary in appearance somewhat depending on the distribution, it will typically include your *username@machinename*, followed by the current working directory (more about that in a little bit) and a dollar sign.

Note: If the last character of the prompt is a pound sign (“#”) rather than a dollar

sign, the terminal session has *superuser* privileges. This means either we are logged in as the root user or we selected a terminal emulator that provides superuser (administrative) privileges.

Assuming things are good so far, let's try some typing. Enter some gibberish at the prompt like so:

```
[me@linuxbox ~]$ kaekfjaeifj
```

Because this command makes no sense, the shell tells us so and give us another chance.

```
bash: kaekfjaeifj: command not found
[me@linuxbox ~]$
```

Command History

If we press the up-arrow key, we will see that the previous command `kaekfjaeifj` reappears after the prompt. This is called *command history*. Most Linux distributions remember the last 1000 commands by default. Press the down-arrow key and the previous command disappears.

Cursor Movement

Recall the previous command by pressing the up-arrow key again. If we try the left and right-arrow keys, we'll see how we can position the cursor anywhere on the command line. This makes editing commands easy.

A Few Words About Mice and Focus

While the shell is all about the keyboard, you can also use a mouse with your terminal emulator. A mechanism built into the X Window System (the underlying engine that makes the GUI go) supports a quick copy and paste technique. If you highlight some text by holding down the left mouse button and dragging the mouse over it (or double clicking on a word), it is copied into a buffer maintained by X. Pressing the middle mouse button will cause the text to be pasted at the cursor location. Try it.

Note: Don't be tempted to use `Ctrl-c` and `Ctrl-v` to perform copy and paste inside a terminal window. They don't work. These control codes have different meanings to the shell and were assigned many years before the release of Microsoft Windows.

Your graphical desktop environment (most likely KDE or GNOME), in an effort to behave like Windows, probably has its *focus policy* set to "click to focus." This means for a window to get focus (become active) you need to click on it. This is contrary to the traditional X behavior of "focus follows mouse" which means that a window gets focus just by passing the mouse over it. The window will not come to the foreground until you click on it but it will be able to receive input. Setting the focus policy to "focus follows mouse" will make the copy and paste technique even more useful. Give it a try if you can (some desktop environments such as Ubuntu's Unity no longer support it). I think if you give it a chance you will prefer it. You will find this setting in the configuration program for your window manager.

Try Some Simple Commands

Now that we have learned to enter text in our terminal emulator, let's try a few simple commands. Let's begin with the `date` command, which displays the current time and date.

```
[me@linuxbox ~]$ date
Thu Mar  8 15:09:41 EST 2018
```

A related command is `cal` which, by default, displays a calendar of the current month.

```
[me@linuxbox ~]$ cal
     March 2018
Su Mo Tu We Th Fr Sa
             1  2  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30 31
```

To see the current amount of free space on our disk drives, enter `df`.

```
[me@linuxbox ~]$ df
Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/sda2             15115452   5012392   9949716  34% /
/dev/sda5             59631908  26545424  30008432  47% /home
/dev/sda1               147764     17370    122765  13% /boot
tmpfs                   256856         0    256856   0% /dev/shm
```

Likewise, to display the amount of free memory, enter the `free` command.

```
[me@linuxbox ~]$ free
             total       used       free     shared    buffers     cached
Mem:    513712     503976       9736          0       5312     122916
-/+ buffers/cache: 375748     137964
Swap:  1052248     104712     947536
```

Ending a Terminal Session

We can end a terminal session by either closing the terminal emulator window, by entering the `exit` command at the shell prompt, or pressing `Ctrl-d`.

```
[me@linuxbox ~]$ exit
```

The Console Behind the Curtain

Even if we have no terminal emulator running, several terminal sessions continue to run behind the graphical desktop., We can access these sessions, called *virtual terminals* or *virtual consoles*, by pressing `Ctrl-Alt-F1` through `Ctrl-Alt-F6` on most Linux distributions. When a session is accessed, it presents a login prompt into which we can enter our username and password. To switch from one virtual console to another, press `Alt-F1` through `Alt-F6`. On most system we can return to the graphical desktop by pressing `Alt-F7`.

Summing Up

This chapter marks the beginning of our journey into the Linux command line with an introduction to the shell and a brief glimpse at the command line and a lesson on how to

start and end a terminal session. We also saw how to issue some simple commands and perform a little light command line editing. That wasn't so scary was it?

In the next chapter, we'll learn a few more commands and wander around the Linux file system.

Further Reading

- To learn more about Steve Bourne, father of the Bourne Shell, see this Wikipedia article:
  http://en.wikipedia.org/wiki/Steve_Bourne
- This Wikipedia article is about Brian Fox, the original author of `bash`:
  https://en.wikipedia.org/wiki/Brian_Fox_(computer_programmer)
- Here is an article about the concept of shells in computing:
  http://en.wikipedia.org/wiki/Shell_(computing)

2 – Navigation

The first thing we need to learn (besides how to type) is how to navigate the file system on our Linux system. In this chapter we will introduce the following commands:

- `pwd` – Print name of current working directory
- `cd` – Change directory
- `ls` – List directory contents

Understanding the File System Tree

Like Windows, a Unix-like operating system such as Linux organizes its files in what is called a *hierarchical directory structure*. This means they are organized in a tree-like pattern of *directories* (sometimes called folders in other systems), which may contain files and other directories. The first directory in the file system is called the *root directory*. The root directory contains files and subdirectories, which contain more files and subdirectories and so on.

Note that unlike Windows, which has a separate file system tree for each storage device, Unix-like systems such as Linux always have a single file system tree, regardless of how many drives or storage devices are attached to the computer. Storage devices are attached (or more correctly, *mounted*) at various points on the tree according to the whims of the *system administrator*, the person (or people) responsible for the maintenance of the system.

The Current Working Directory

Most of us are probably familiar with a graphical file manager which represents the file system tree as in Figure 1. Notice that the tree is usually shown upended, that is, with the root at the top and the various branches descending below.

However, the command line has no pictures, so to navigate the file system tree we need to think of it in a different way.

Imagine that the file system is a maze shaped like an upside-down tree and we are able to

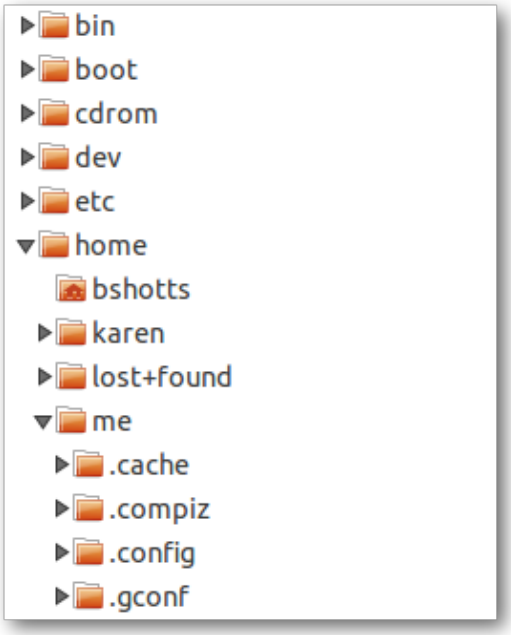

Figure 1: File system tree as shown by a graphical file manager

stand in the middle of it. At any given time, we are inside a single directory and we can see the files contained in the directory and the pathway to the directory above us (called the *parent directory*) and any subdirectories below us. The directory we are standing in is called the *current working directory*. To display the current working directory, we use the `pwd` (print working directory) command.

```
[me@linuxbox ~]$ pwd
/home/me
```

When we first log in to our system (or start a terminal emulator session) our current working directory is set to our *home directory*. Each user account is given its own home directory and it is the only place a regular user is allowed to write files.

Listing the Contents of a Directory

To list the files and directories in the current working directory, we use the `ls` command.

```
[me@linuxbox ~]$ ls
```

```
Desktop  Documents  Music  Pictures  Public  Templates  Videos
```

Actually, we can use the `ls` command to list the contents of any directory, not just the current working directory, and there are many other fun things it can do as well. We'll spend more time with `ls` in the next chapter.

Changing the Current Working Directory

To change our working directory (where we are standing in our tree-shaped maze) we use the `cd` command. To do this, type `cd` followed by the *pathname* of the desired working directory. A pathname is the route we take along the branches of the tree to get to the directory we want. We can specify pathnames in one of two different ways; as *absolute pathnames* or as *relative pathnames*. Let's deal with absolute pathnames first.

Absolute Pathnames

An absolute pathname begins with the root directory and follows the tree branch by branch until the path to the desired directory or file is completed. For example, there is a directory on our system in which most of our system's programs are installed. The directory's pathname is `/usr/bin`. This means from the root directory (represented by the leading slash in the pathname) there is a directory called "usr" which contains a directory called "bin".

```
[me@linuxbox ~]$ cd /usr/bin
[me@linuxbox bin]$ pwd
/usr/bin
[me@linuxbox bin]$ ls
...Listing of many, many files ...
```

Now we can see that we have changed the current working directory to `/usr/bin` and that it is full of files. Notice how the shell prompt has changed? As a convenience, it is usually set up to automatically display the name of the working directory.

Relative Pathnames

Where an absolute pathname starts from the root directory and leads to its destination, a relative pathname starts from the working directory. To do this, it uses a couple of special notations to represent relative positions in the file system tree. These special notations are "." (dot) and ".." (dot dot).

The "." notation refers to the working directory and the ".." notation refers to the working

directory's parent directory. Here is how it works. Let's change the working directory to `/usr/bin` again.

```
[me@linuxbox ~]$ cd /usr/bin
[me@linuxbox bin]$ pwd
/usr/bin
```

Now let's say that we wanted to change the working directory to the parent of `/usr/bin` which is `/usr`. We could do that two different ways, either using an absolute pathname.

```
[me@linuxbox bin]$ cd /usr
[me@linuxbox usr]$ pwd
/usr
```

or, using a relative pathname.

```
[me@linuxbox bin]$ cd ..
[me@linuxbox usr]$ pwd
/usr
```

Two different methods with identical results. Which one should we use? The one that requires the least typing!

Likewise, we can change the working directory from `/usr` to `/usr/bin` in two different ways, either using an absolute pathname:

```
[me@linuxbox usr]$ cd /usr/bin
[me@linuxbox bin]$ pwd
/usr/bin
```

or, using a relative pathname.

```
[me@linuxbox usr]$ cd ./bin
[me@linuxbox bin]$ pwd
/usr/bin
```

Now, there is something important to point out here. In almost all cases, we can omit the "./". It is implied. Typing:

```
[me@linuxbox usr]$ cd bin
```

does the same thing. In general, if we do not specify a pathname to something, the working directory will be assumed.

Some Helpful Shortcuts

In Table 2-1 we see some useful ways the current working directory can be quickly changed.

Table 2-1: `cd` Shortcuts

Shortcut	Result
`cd`	Changes the working directory to your home directory.
`cd -`	Changes the working directory to the previous working directory.
`cd ~`*`user_name`*	Changes the working directory to the home directory of *user_name*. For example, `cd ~bob` will change the directory to the home directory of user “bob.”

Important Facts About Filenames

On Linux systems, files are named in a manner similar to other systems such as Windows, but there are some important differences.

1. Filenames that begin with a period character are hidden. This only means that `ls` will not list them unless you say `ls -a`. When your account was created, several hidden files were placed in your home directory to configure things for your account. In Chapter 11 we will take a closer look at some of these files to see how you can customize your environment. In addition, some applications place their configuration and settings files in your home directory as hidden files.

2. Filenames and commands in Linux, like Unix, are case sensitive. The filenames “File1” and “file1” refer to different files.
3. Linux has no concept of a “file extension” like some other operating systems. You may name files any way you like. The contents and/or purpose of a file is determined by other means. Although Unix-like operating systems don’t use file extensions to determine the contents/purpose of files, many application programs do.
4. Though Linux supports long filenames that may contain embedded spaces and punctuation characters, limit the punctuation characters in the names of files you create to period, dash, and underscore. *Most importantly, do not embed spaces in filenames.* If you want to represent spaces between words in a filename, use underscore characters. You will thank yourself later.

Summing Up

This chapter explained how the shell treats the directory structure of the system. We learned about absolute and relative pathnames and the basic commands that we use to move about that structure. In the next chapter we will use this knowledge to go on a tour of a modern Linux system.

3 – Exploring the System

Now that we know how to move around the file system, it's time for a guided tour of our Linux system. Before we start however, we're going to learn some more commands that will be useful along the way.

- `ls` – List directory contents
- `file` – Determine file type
- `less` – View file contents

Having More Fun with `ls`

The `ls` command is probably the most used command, and for good reason. With it, we can see directory contents and determine a variety of important file and directory attributes. As we have seen, we can simply enter `ls` to get a list of files and subdirectories contained in the current working directory.

```
[me@linuxbox ~]$ ls
Desktop  Documents  Music  Pictures  Public  Templates  Videos
```

Besides the current working directory, we can specify the directory to list, like so:

```
me@linuxbox ~]$ ls /usr
bin  games  include  lib  local  sbin  share  src
```

We can even specify multiple directories. In the following example, we list both the user's home directory (symbolized by the "~" character) and the `/usr` directory.

```
[me@linuxbox ~]$ ls ~ /usr
/home/me:
Desktop  Documents  Music  Pictures  Public  Templates  Videos
```

```
/usr:
bin  games  include  lib  local  sbin  share  src
```

We can also change the format of the output to reveal more detail.

```
[me@linuxbox ~]$ ls -l
total 56
drwxrwxr-x 2 me me 4096 2017-10-26 17:20 Desktop
drwxrwxr-x 2 me me 4096 2017-10-26 17:20 Documents
drwxrwxr-x 2 me me 4096 2017-10-26 17:20 Music
drwxrwxr-x 2 me me 4096 2017-10-26 17:20 Pictures
drwxrwxr-x 2 me me 4096 2017-10-26 17:20 Public
drwxrwxr-x 2 me me 4096 2017-10-26 17:20 Templates
drwxrwxr-x 2 me me 4096 2017-10-26 17:20 Videos
```

By adding “-l” to the command, we changed the output to the long format.

Options and Arguments

This brings us to a very important point about how most commands work. Commands are often followed by one or more *options* that modify their behavior, and further, by one or more *arguments*, the items upon which the command acts. So most commands look kind of like this:

```
command -options arguments
```

Most commands use options which consist of a single character preceded by a dash, for example, “-l”. Many commands, however, including those from the GNU Project, also support *long options*, consisting of a word preceded by two dashes. Also, many commands allow multiple short options to be strung together. In the following example, the `ls` command is given two options, which are the `l` option to produce long format output, and the `t` option to sort the result by the file's modification time.

```
[me@linuxbox ~]$ ls -lt
```

We'll add the long option “--reverse” to reverse the order of the sort.

```
[me@linuxbox ~]$ ls -lt --reverse
```

> Note that command options, like filenames in Linux, are case-sensitive.

The `ls` command has a large number of possible options. The most common are listed in Table 3-1.

Table 3- 1: Common `ls` Options

Option	Long Option	Description
`-a`	`--all`	List all files, even those with names that begin with a period, which are normally not listed (that is, hidden).
`-A`	`--almost-all`	Like the `-a` option above except it does not list `.` (current directory) and `..` (parent directory).
`-d`	`--directory`	Ordinarily, if a directory is specified, `ls` will list the contents of the directory, not the directory itself. Use this option in conjunction with the `-l` option to see details about the directory rather than its contents.
`-F`	`--classify`	This option will append an indicator character to the end of each listed name. For example, a forward slash (/) if the name is a directory.
`-h`	`--human-readable`	In long format listings, display file sizes in human readable format rather than in bytes.
`-l`		Display results in long format.
`-r`	`--reverse`	Display the results in reverse order. Normally, `ls` displays its results in ascending alphabetical order.
`-S`		Sort results by file size.
`-t`		Sort by modification time.

A Longer Look at Long Format

As we saw earlier, the `-l` option causes `ls` to display its results in long format. This format contains a great deal of useful information. Here is the `Examples` directory from an Ubuntu system:

```
-rw-r--r-- 1 root root 3576296 2017-04-03 11:05 Experience ubuntu.ogg
-rw-r--r-- 1 root root 1186219 2017-04-03 11:05 kubuntu-leaflet.png
-rw-r--r-- 1 root root   47584 2017-04-03 11:05 logo-Edubuntu.png
-rw-r--r-- 1 root root   44355 2017-04-03 11:05 logo-Kubuntu.png
-rw-r--r-- 1 root root   34391 2017-04-03 11:05 logo-Ubuntu.png
-rw-r--r-- 1 root root   32059 2017-04-03 11:05 oo-cd-cover.odf
-rw-r--r-- 1 root root  159744 2017-04-03 11:05 oo-derivatives.doc
-rw-r--r-- 1 root root   27837 2017-04-03 11:05 oo-maxwell.odt
-rw-r--r-- 1 root root   98816 2017-04-03 11:05 oo-trig.xls
-rw-r--r-- 1 root root  453764 2017-04-03 11:05 oo-welcome.odt
-rw-r--r-- 1 root root  358374 2017-04-03 11:05 ubuntu Sax.ogg
```

Table 3-2 provides us with a look at the different fields from one of the files and their meanings.

Table 3-2: `ls` Long Listing Fields

Field	Meaning
`-rw-r--r--`	Access rights to the file. The first character indicates the type of file. Among the different types, a leading dash means a regular file, while a “d” indicates a directory. The next three characters are the access rights for the file's owner, the next three are for members of the file's group, and the final three are for everyone else. Chapter 9 "Permissions" discusses the full meaning of this in more detail.
`1`	File's number of hard links. See the sections "Symbolic Links" and "Hard Links" later in this chapter.
`root`	The username of the file's owner.
`root`	The name of the group that owns the file.
`32059`	Size of the file in bytes.
`2007-04-03 11:05`	Date and time of the file's last modification.
`oo-cd-cover.odf`	Name of the file.

Determining a File's Type with `file`

As we explore the system it will be useful to know what files contain. To do this we will use the `file` command to determine a file's type. As we discussed earlier, filenames in Linux are not required to reflect a file's contents. While a filename like “picture.jpg” would normally be expected to contain a JPEG compressed image, it is not required to in Linux. We can invoke the `file` command this way:

```
file filename
```

When invoked, the `file` command will print a brief description of the file's contents. For example:

```
[me@linuxbox ~]$ file picture.jpg
picture.jpg: JPEG image data, JFIF standard 1.01
```

There are many kinds of files. In fact, one of the common ideas in Unix-like operating systems such as Linux is that “everything is a file.” As we proceed with our lessons, we will see just how true that statement is.

While many of the files on our system are familiar, for example MP3 and JPEG, there are many kinds that are a little less obvious and a few that are quite strange.

Viewing File Contents with `less`

The `less` command is a program to view text files. Throughout our Linux system, there are many files that contain human-readable text. The `less` program provides a convenient way to examine them.

What Is “Text”?

There are many ways to represent information on a computer. All methods involve defining a relationship between the information and some numbers that will be used to represent it. Computers, after all, only understand numbers and all data is converted to numeric representation.

Some of these representation systems are very complex (such as compressed video files), while others are rather simple. One of the earliest and simplest is called *ASCII text*. ASCII (pronounced "As-Key") is short for American Standard Code for Information Interchange. This is a simple encoding scheme that was first used on Teletype machines to map keyboard characters to numbers.

Text is a simple one-to-one mapping of characters to numbers. It is very compact. Fifty characters of text translates to fifty bytes of data. It is important to understand that text only contains a simple mapping of characters to numbers. It is not the same as a word processor document such as one created by Microsoft Word or LibreOffice Writer. Those files, in contrast to simple ASCII text, contain many non-text elements that are used to describe its structure and formatting. Plain ASCII text files contain only the characters themselves and a few rudimentary control codes such as tabs, carriage returns and line feeds.

Throughout a Linux system, many files are stored in text format and there are many Linux tools that work with text files. Even Windows recognizes the importance of this format. The well-known NOTEPAD.EXE program is an editor for plain ASCII text files.

Why would we want to examine text files? Because many of the files that contain system settings (called *configuration files*) are stored in this format, and being able to read them gives us insight about how the system works. In addition, some of the actual programs that the system uses (called *scripts*) are stored in this format. In later chapters, we will learn how to edit text files in order to modify systems settings and write our own scripts, but for now we will just look at their contents.

The `less` command is used like this:

```
less filename
```

Once started, the `less` program allows us to scroll forward and backward through a text file. For example, to examine the file that defines all the system's user accounts, enter the following command:

```
[me@linuxbox ~]$ less /etc/passwd
```

Once the `less` program starts, we can view the contents of the file. If the file is longer

than one page, we can scroll up and down. To exit `less`, press the `q` key.

The table below lists the most common keyboard commands used by `less`.

Table 3-3: `less` Commands

Command	Action
Page Up or `b`	Scroll back one page
Page Down or space	Scroll forward one page
Up arrow	Scroll up one line
Down arrow	Scroll down one line
`G`	Move to the end of the text file
`1G` or `g`	Move to the beginning of the text file
`/`*characters*	Search forward to the next occurrence of *characters*
`n`	Search for the next occurrence of the previous search
`h`	Display help screen
`q`	Quit `less`

Less Is More

The `less` program was designed as an improved replacement of an earlier Unix program called `more`. The name "less" is a play on the phrase "less is more"—a motto of modernist architects and designers.

`less` falls into the class of programs called "pagers," programs that allow the easy viewing of long text documents in a page by page manner. Whereas the `more` program could only page forward, the `less` program allows paging both forward and backward and has many other features as well.

Taking a Guided Tour

The file system layout on a Linux system is much like that found on other Unix-like systems. The design is actually specified in a published standard called the *Linux Filesystem Hierarchy Standard*. Not all Linux distributions conform to the standard exactly but most come pretty close.

Next, we are going to wander around the file system ourselves to see what makes our Linux system tick. This will give us a chance to practice our navigation skills. One of the things we will discover is that many of the interesting files are in plain human-readable text. As we go about our tour, try the following:

1. `cd` into a given directory
2. List the directory contents with `ls -l`
3. If you see an interesting file, determine its contents with `file`
4. If it looks like it might be text, try viewing it with `less`
5. If we accidentally attempt to view a non-text file and it scrambles the terminal window, we can recover by entering the `reset` command.

Remember the copy and paste trick! If you are using a mouse, you can double click on a filename to copy it and middle click to paste it into commands.

As we wander around, don't be afraid to look at stuff. Regular users are largely prohibited from messing things up. That's the system administrator's job! If a command complains about something, just move on to something else. Spend some time looking around. The system is ours to explore. Remember, in Linux, there are no secrets!

Table 3-4 lists just a few of the directories we can explore. There may be some slight differences depending on our Linux distribution. Don't be afraid to look around and try more!

Table 3-4: Directories Found on Linux Systems

Directory	Comments
`/`	The root directory. Where everything begins.
`/bin`	Contains binaries (programs) that must be present for the system to boot and run.
`/boot`	Contains the Linux kernel, initial RAM disk image (for drivers needed at boot time), and the boot loader. Interesting files: • `/boot/grub/grub.conf` or `menu.lst`, which are used to configure the boot loader. • `/boot/vmlinuz` (or something similar), the Linux kernel

Directory	Comments
`/dev`	This is a special directory that contains *device nodes*. "Everything is a file" also applies to devices. Here is where the kernel maintains a list of all the devices it understands.
`/etc`	The `/etc` directory contains all of the system-wide configuration files. It also contains a collection of shell scripts that start each of the system services at boot time. Everything in this directory should be readable text. Interesting files: While everything in `/etc` is interesting, here are some all-time favorites: • `/etc/crontab`, a file that defines when automated jobs will run. • `/etc/fstab`, a table of storage devices and their associated mount points. • `/etc/passwd`, a list of the user accounts.
`/home`	In normal configurations, each user is given a directory in `/home`. Ordinary users can only write files in their home directories. This limitation protects the system from errant user activity.
`/lib`	Contains shared library files used by the core system programs. These are similar to dynamic link libraries (DLLs) in Windows.
`/lost+found`	Each formatted partition or device using a Linux file system, such as ext4, will have this directory. It is used in the case of a partial recovery from a file system corruption event. Unless something really bad has happened to our system, this directory will remain empty.
`/media`	On modern Linux systems the `/media` directory will contain the mount points for removable media such as USB drives, CD-ROMs, etc. that are mounted automatically at insertion.
`/mnt`	On older Linux systems, the `/mnt` directory contains mount points for removable devices that have been mounted manually.

Directory	Comments
/opt	The /opt directory is used to install “optional” software. This is mainly used to hold commercial software products that might be installed on the system.
/proc	The /proc directory is special. It's not a real file system in the sense of files stored on the hard drive. Rather, it is a virtual file system maintained by the Linux kernel. The “files” it contains are peepholes into the kernel itself. The files are readable and will give us a picture of how the kernel sees the computer.
/root	This is the home directory for the root account.
/sbin	This directory contains “system” binaries. These are programs that perform vital system tasks that are generally reserved for the superuser.
/tmp	The /tmp directory is intended for the storage of temporary, transient files created by various programs. Some configurations cause this directory to be emptied each time the system is rebooted.
/usr	The /usr directory tree is likely the largest one on a Linux system. It contains all the programs and support files used by regular users.
/usr/bin	/usr/bin contains the executable programs installed by the Linux distribution. It is not uncommon for this directory to hold thousands of programs.
/usr/lib	The shared libraries for the programs in /usr/bin.
/usr/local	The /usr/local tree is where programs that are not included with the distribution but are intended for system-wide use are installed. Programs compiled from source code are normally installed in /usr/local/bin. On a newly installed Linux system, this tree exists, but it will be empty until the system administrator puts something in it.
/usr/sbin	Contains more system administration programs.
/usr/share	/usr/share contains all the shared data used by programs in /usr/bin. This includes things such as default configuration files, icons, screen backgrounds, sound files, etc.

Directory	Comments
`/usr/share/doc`	Most packages installed on the system will include some kind of documentation. In `/usr/share/doc`, we will find documentation files organized by package.
`/var`	With the exception of `/tmp` and `/home`, the directories we have looked at so far remain relatively static, that is, their contents don't change. The `/var` directory tree is where data that is likely to change is stored. Various databases, spool files, user mail, etc. are located here.
`/var/log`	`/var/log` contains *log files*, records of various system activity. These are important and should be monitored from time to time. The most useful ones are `/var/log/messages` and `/var/log/syslog`. Note that for security reasons on some systems only the superuser may view log files.

Symbolic Links

As we look around, we are likely to see a directory listing (for example, `/lib`) with an entry like this:

```
lrwxrwxrwx 1 root root   11 2007-08-11 07:34 libc.so.6 -> libc-2.6.so
```

Notice how the first letter of the listing is “l” and the entry seems to have two filenames? This is a special kind of a file called a *symbolic link* (also known as a *soft link* or *sym-link*). In most Unix-like systems it is possible to have a file referenced by multiple names. While the value of this might not be obvious, it is really a useful feature.

Picture this scenario: A program requires the use of a shared resource of some kind contained in a file named “foo,” but “foo” has frequent version changes. It would be good to include the version number in the filename so the administrator or other interested party could see what version of “foo” is installed. This presents a problem. If we change the name of the shared resource, we have to track down every program that might use it and change it to look for a new resource name every time a new version of the resource is installed. That doesn't sound like fun at all.

Here is where symbolic links save the day. Suppose we install version 2.6 of “foo,” which has the filename “foo-2.6” and then create a symbolic link simply called “foo” that points to “foo-2.6.” This means that when a program opens the file “foo”, it is actually opening the file “foo-2.6”. Now everybody is happy. The programs that rely on “foo” can find it

and we can still see what actual version is installed. When it is time to upgrade to "foo-2.7," we just add the file to our system, delete the symbolic link "foo" and create a new one that points to the new version. Not only does this solve the problem of the version upgrade, but it also allows us to keep both versions on our machine. Imagine that "foo-2.7" has a bug (damn those developers!) and we need to revert to the old version. Again, we just delete the symbolic link pointing to the new version and create a new symbolic link pointing to the old version.

The directory listing at the beginning of this section (from the `/lib` directory of a Fedora system) shows a symbolic link called `libc.so.6` that points to a shared library file called `libc-2.6.so`. This means that programs looking for `libc.so.6` will actually get the file `libc-2.6.so`. We will learn how to create symbolic links in the next chapter.

Hard Links

While we are on the subject of links, we need to mention that there is a second type of link called a *hard link*. Hard links also allow files to have multiple names, but they do it in a different way. We'll talk more about the differences between symbolic and hard links in the next chapter.

Summing Up

With our tour behind us, we have learned a lot about our system. We've seen various files and directories and their contents. One thing we should take away from this is how open the system is. In Linux there are many important files that are plain human-readable text. Unlike many proprietary systems, Linux makes everything available for examination and study.

Further Reading

- The full version of the *Linux Filesystem Hierarchy Standard* can be found here: http://www.pathname.com/fhs/
- An article about the directory structure of Unix and Unix-like systems: http://en.wikipedia.org/wiki/Unix_directory_structure
- A detailed description of the ASCII text format: http://en.wikipedia.org/wiki/ASCII

4 – Manipulating Files and Directories

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

- `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

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

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, and so on. So why use these old command line programs?

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? It's pretty hard with a file manager but pretty easy with the command line.

```
cp -u *.html destination
```

Wildcards

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 us rapidly specify groups of filenames. These special characters are

called *wildcards*. Using wildcards (which is also known as *globbing*) allows us to select filenames based on patterns of characters. Table 4-1 lists the wildcards and what they select.

Table 4-1: 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*

Table 4-2 lists the most commonly used character classes.

Table 4-2: 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

Using wildcards makes it possible to construct sophisticated selection criteria for filenames. Table 4-3 provides some examples of patterns and what they match.

Table 4-3: Wildcard Examples

Pattern	Matches
`*`	All files
`g*`	Any 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”

Wildcards can be used with any command that accepts filenames as arguments, but we’ll talk more about that in Chapter 7, "Seeing the World As the Shell Sees It.

Character Ranges

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]` and `[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 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.

Wildcards Work in the GUI Too

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

- 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 some versions of 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, enter "/usr/bin/u*" in the location bar and it will display the result.

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.

`mkdir` – Create Directories

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

```
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 the following command:

```
mkdir dir1
```

would create a single directory named `dir1`, while the following:

```
mkdir dir1 dir2 dir3
```

would create three directories named `dir1`, `dir2`, and `dir3`.

`cp` – Copy Files and Directories

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

```
cp item1 item2
```

copies the single file or directory `item1` to the file or directory `item2` and the following:

```
cp item... directory
```

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

Useful Options and Examples

Table 4-4 lists some of the commonly used options for `cp`.

Table 4-4: `cp` Options

Option	Long 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. We'll take a look at file permissions in Chapter 9 "Permissions."
`-i`	`--interactive`	Before overwriting an existing file, prompt the user for confirmation. **If this option is not specified, `cp` will silently (meaning there will be no warning) 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. This is useful when copying large numbers of files as it skips files that don't need to be copied.
`-v`	`--verbose`	Display informative messages as the copy is performed.

Table 4-5: `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 previous command, except that if *file2* exists, the user is prompted before it is overwritten.
`cp file1 file2 dir1`	Copy *file1* and *file2* into directory *dir1*. The directory *dir1* must already exist.
`cp dir1/* dir2`	Using a wildcard, copy all the files in *dir1* into *dir2*. The directory *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*.

`mv` – Move and Rename Files

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`, as shown here:

```
mv item1 item2
```

to move or rename the file or directory `item1` to `item2` or:

```
mv item... directory
```

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

Useful Options and Examples

`mv` shares many of the same options as `cp` as described in Table 4-6.

Table 4-6: `mv` Options

Option	Long Option	Meaning
`-i`	`--interactive`	Before overwriting an existing file, prompt the

		user for confirmation. **If this option is not specified, `mv` 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.

Table 4-7 provides some examples of `mv` usage.

Table 4-7: `mv` Examples

Command	Results
`mv` *`file1 file2`*	Move *file1* to *file2*. **If *file2* exists, it is overwritten with the contents of *file1*.** If *file2* does not exist, it is created. **In either case, *file1* ceases to exist.**
`mv -i` *`file1 file2`*	Same as the previous command, except that if *file2* exists, the user is prompted before it is overwritten.
`mv` *`file1 file2 dir1`*	Move *file1* and *file2* into directory *dir1*. The dierctory *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*.

`rm` – Remove Files and Directories

The `rm` command is used to remove (delete) files and directories, as shown here:

```
rm item...
```

where `item` is one or more files or directories.

Useful Options and Examples

Table 4-8 describes some of the common options for `rm`.

Table 4-8: `rm` Options

Option	Long 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.

Table 4-9 provides some examples of using the `rm` command.

Table 4-9: `rm` Examples

Command	Results
`rm` *`file1`*	Delete *file1* silently.
`rm -i` *`file1`*	Same as the previous command, 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 the previous command, except that if either *file1* or *dir1* do not exist, `rm` will continue silently.