The shell

So you've installed Slackware and you're staring at a terminal prompt, what now? Now would be a good time to learn about the basic command line tools. And since you're staring at a blinking curser, you probably need a little assistance in knowing how to get around, and that is what this chapter is all about.

System Documentation

Your Slackware Linux system comes with lots of built-in documentation for nearly every installed application. Perhaps the most common method of reading system documentation is ***man***(1). ***man*** (short for **manual**) will bring up the included man-page for any application, system call, configuration file, or library you tell it too. For example, **man man** will bring up the man-page for ***man*** itself.

Unfortunately, you may not always know what application you need to use for the task at hand. Thankfully, ***man*** has built-in search abilities. Using the *-k* switch will cause ***man*** to search for every man-page that matches your search terms.

The man-pages are organized into groups or sections by their content type. For example, section 1 is for user applications. ***man*** will search each section in order and display the first match it finds. Sometimes you will find that a man-page exists in more than one section for a given entry. In that case, you will need to specify the exact section to look in. In this book, all applications and a number of other things will have a number on their right-hand side in parenthesis. This number is the man page section where you will find information on that tool.

```
darkstar:~$ man -k printf
printf               (1)  - format and print data
printf               (3)  - formatted output conversion
darkstar:~$ man 3 printf
```

Man Page Sections

Section	Contents
1	User Commands
2	System Calls
3	C Library Calls
4	Devices
5	File Formats / Protocols
6	Games
7	Conventions / Macro Packages
8	System Administration
9	Kernel API Descriptions

Section	Contents
n	“New” - typically used to Tcl/Tk

Dealing with Files and Directories

Listing Files and Directory Contents

ls(1) is used to list files and directories, their permissions, size, type, inode number, owner and group, and plenty of additional information. For example, let's list what's in the / directory for your new Slackware Linux system.

```
darkstar:~$ ls /
bin/   dev/  home/  lost+found/  mnt/  proc/  sbin/  sys/  usr/
boot/  etc/  lib/   media/       opt/  root/  srv/   tmp/  var/
```

Notice that each of the listings is a directory. These are easily distinguished from regular files due to the trailing /; standard files do not have a suffix. Additionally, executable files will have an asterisk suffix. But ***ls*** can do so much more. To get a view of the permissions of a file or directory, you'll need to do a *“long list”*.

```
darkstar:~$ ls -l /home/alan/Desktop
-rw-r--r-- 1 alan users 15624161 2007-09-21 13:02 9780596510480.pdf
-rw-r--r-- 1 alan users  3829534 2007-09-14 12:56 imgscan.zip
drwxr-xr-x 3 alan root       168 2007-09-17 21:01 ipod_hack/
drwxr-xr-x 2 alan users      200 2007-12-03 22:11 libgpod/
drwxr-xr-x 2 alan users      136 2007-09-30 03:16 playground/
```

A long listing lets you view the permisions, user and group ownership, file size, last modified date, and of course, the file name itself. Notice that the first two entires are files, and the last three are directories. This is denoted by the very first character on the line. Regular files get a *“-”*; directories get a *“d”*. There are several other file types with their own denominators. Symbolic links for example will have an *“l”*.

Lastly, we'll show you how to list dot-files, or hidden files. Unlike other operating systems such as Microsoft Windows, there is no special property that differentiates *“hidden”* files from *“unhidden”* files. A hidden file simply begins with a dot. To display these files along with all the others, you just need to pass the *-a* argument to ***ls***.

```
darkstar:~$ ls -a
.xine/    .xinitrc-backup  .xscreensaver  .xsession-errors  SBo/
.xinitrc  .xinitrc-xfce    .xsession      .xwmconfig/       Shared/
```

You also likely noticed that your files and directories appear in different colors. Many of the enhanced features of ***ls*** such as these colors or the trailing characters indicating file-type are special features of the ***ls*** program that are turned on by passing various arguments. As a convienience, Slackware sets up ***ls*** to use many of these optional arguments by default. These are controlled by the LS_OPTIONS and LS_COLORS environment variables. We will talk more about environment variables in chapter 5.

Moving Around the Filesystem

cd is the command used to change directories. Unlike most other commands, ***cd*** is actually not it's own program, but is a shell built-in. Basically, that means ***cd*** does not have its own man page. You'll have to check your shell's documentation for more details on the ***cd*** you may be using. For the most part though, they all behave the same.

```
darkstar:~$ cd /
darkstar:/$ls
bin/   dev/  home/  lost+found/  mnt/  proc/  sbin/  sys/  usr/
boot/  etc/  lib/   media/       opt/  root/  srv/   tmp/  var/
darkstar:/$cd /usr/local
darkstar:/usr/local$
```

Notice how the prompt changed when we changed directories? The default Slackware shell does this as a quick, easy way to see your current directory, but this is not actually a function of ***cd***. If your shell doesn't operate in this way, you can easily get your current working directory with the ***pwd***(1) command. (Most UNIX shells have configurable prompts that can be coaxed into providing this same functionality. In fact, this is another convience setup in the default shell for you by Slackware.)

```
darkstar:~$ pwd
/usr/local
```

File and Directory Creation and Deletion

While most applications can and will create their own files and directories, you'll often want to do this on your own. Thankfully, it's very easy using ***touch***(1) and ***mkdir***(1).

touch actually modifies the timestamp on a file, but if that file doesn't exist, it will be created.

```
darkstar:~/foo$ ls -l
-rw-r--r-- 1 alan users 0 2012-01-18 15:01 bar1
darkstar:~/foo$ touch bar2
-rw-r--r-- 1 alan users 0 2012-01-18 15:01 bar1
-rw-r--r-- 1 alan users 0 2012-01-18 15:05 bar2
darkstar:~/foo$ touch bar1
-rw-r--r-- 1 alan users 0 2012-01-18 15:05 bar1
-rw-r--r-- 1 alan users 0 2012-01-18 15:05 bar2
```

Note how `bar2` was created in our second command, and the third command simpl updated the timestamp on `bar1`

mkdir is used for (obviously enough) making directories. mkdir foo will create the directory “foo” within the current working directory. Additionally, you can use the *-p* argument to create any missing parent directories.

```
darkstar:~$ mkdir foo
```

```
darkstar:~$ mkdir /slack/foo/bar/
mkdir: cannot create directory `/slack/foo/bar/': No such file or directory
darkstar:~$ mkdir -p /slack/foo/bar/
```

In the latter case, ***mkdir*** will first create “/slack”, then “/slack/foo”, and finally “/slack/foo/bar”. If you failed to use the *-p* argument, ***mkdir*** would fail to create “/slack/foo/bar” unless the first two already existed, as you saw in the example.

Removing a file is as easy as creating one. The ***rm***(1) command will remove a file (assuming of course that you have permission to do this). There are a few very common arguments to ***rm***. The first is *-f* and is used to force the removal of a file that you may lack explicit permission to delete. The *-r* argument will remove directories and their contents recursively.

There is another tool to remove directories, the humble ***rmdir***(1). ***rmdir*** will only remove directories that are empty, and complain noisely about those that contain files or sub-directories.

```
darkstar:~$ ls
foo_1/ foo_2/
darkstar:~$ ls foo_1
bar_1
darkstar:~$ rmdir foo_1
rmdir: foo/: Directory not empty
darkstar:~$ rm foo_1/bar
darkstar:~$ rmdir foo_1
darkstar:~$ ls foo_2
bar_2/
darkstar:~$ rm -fr foo_2
darkstar:~$ ls
```

Archive and Compression

Everyone needs to package a lot of small files together for easy storage from time to time, or perhaps you need to compress very large files into a more manageable size? Maybe you want to do both of those together? Thankfully there are several tools to do just that.

zip and unzip

You're probably familiar with .zip files. These are compressed files that contain other files and directories. While we don't normally use these files in the Linux world, they are still commonly used by other operating systems, so we occasionally have to deal with them.

In order to create a zip file, you'll (naturally) use the ***zip***(1) command. You can compress either files or directories (or both) with ***zip***, but you'll have to use the *-r* argument for recursive action in order to deal with directories.

```
darkstar:~$ zip -r /tmp/home.zip /home
darkstar:~$ zip /tmp/large_file.zip /tmp/large_file
```

The order of the arguments is very important. The first filename must be the zip file to create (if the .zip extension is ommitted, ***zip*** will add it for you) and the rest are files or directories to be added to the zip file.

Naturally, ***unzip***(1) will decompress a zip archive file.

```
darkstar:~$ unzip /tmp/home.zip
```

gzip

One of the oldest compression tools included in Slackware is ***gzip***(1), a compression tool that is only capable or operating on a single file at a time. Whereas ***zip*** is both a compression and an archival tool, ***gzip*** is only capable of compression. At first glance this seems like a draw-back, but it is really a strength. The UNIX philosophy of making small tools that do their small jobs well allows them to be combined in myriad ways. In order to compress a file (or multiple files), simply pass them as arguments to ***gzip***. Whenever ***gzip*** compresses a file, it adds a .gz extension and removes the original file.

```
darkstar:~$ gzip /tmp/large_file
```

Decompressing is just as straight-forward with ***gunzip*** which will create a new uncompressed file and delete the old one.

```
darkstar:~$ gunzip /tmp/large_file.gz
darkstar:~$ ls /tmp/large_file*
/tmp/large_file
```

But suppose we don't want to delete the old compressed file, we just want to read its contents or send them as input to another program? The ***zcat*** program will read the gzip file, decompress it in memory, and send the contents to the standard output (the terminal screen unless it is redirected, see the section called “Input and Output Redirection” for more details on output redirection).

```
darkstar:~$ zcat /tmp/large_file.gz
Wed Aug 26 10:00:38 CDT 2009
Slackware 13.0 x86 is released as stable!  Thanks to everyone who helped
make this release possible -- see the RELEASE_NOTES for the credits.
The ISOs are off to the replicator.  This time it will be a 6 CD-ROM
32-bit set and a dual-sided 32-bit/64-bit x86/x86_64 DVD.  We're taking
pre-orders now at store.slackware.com.  Please consider picking up a copy
to help support the project.  Once again, thanks to the entire Slackware
community for all the help testing and fixing things and offering
suggestions during this development cycle.
```

bzip2

One alternative to ***gzip*** is the ***bzip2***(1) compression utility which works in almost the exact same way. The advantage to ***bzip2*** is that it boasts greater compression strength. Unfortunately, achieving that greater compression is a slow and CPU-intensive process, so ***bzip2*** typically takes

much longer to run than other alternatives.

XZ / LZMA

The latest compression utility added to Slackware is ***xz***, which implements the LZMA compression algorithm. This is faster than ***bzip2*** and often compresses better as well. In fact, its blend of speed and compression strength caused it to replace ***gzip*** as the compression scheme of choice for Slackware. Unfortunately, ***xz*** does not have a man page at the time of this writing, so to view available options, use the *–help* argument. Compressing files is accomplished with the *-z* argument, and decompression with *-d*.

```
darkstar:~$ xz -z /tmp/large_file
```

tar

So great, we know how to compress files using all sorts of programs, but none of them can archive files in the way that ***zip*** does. That is until now. The Tape Archiver, or ***tar***(1) is the most frequently used archival program in Slackware. Like other archival programs, ***tar*** generates a new file that contains other files and directories. It does not compress the generated file (often called a "tarball") by default; however, the version of ***tar*** included in Slackware supports a variety of compression schemes, including the ones mentioned above.

Invoking ***tar*** can be as easy or as complicated as you like. Typically, creating a tarball is done with the *-cvzf* arguments. Let's look at these in depth.

tar Arguments

Argument	Meaning
c	Create a tarball
x	Extract the contents of a tarball
t	Display the contents of a tarball
v	Be more verbose
z	Use gzip compression
j	Use bzip2 compression
J	Use LZMA compression
p	Preserve permissions

tar requires a bit more precision than other applications in the order of its arguments. The *-f* argument must be present when reading or writing to a file for example, and the very next thing to follow must be the filename. Consider the following examples.

```
darkstar:~$ tar -xvzf /tmp/tarball.tar.gz
darkstar:~$ tar -xvfz /tmp/tarball.tar.gz
```

Above, the first example works as you would expect, but the second fails because ***tar*** has been instructed to open the `z` file rather than the expected `/tmp/tarball.tar.gz`.

Now that we've got our arguments straightened out, lets look at a few examples of how to create and extract tarballs. As we've noted, the *-c* argument is used to create tarballs and *-x* extracts their contents. If we want to create or extract a compressed tarball though, we also have to specify the proper compression to use. Naturally, if we don't want to compress the tarball at all, we can leave these options out. The following command creates a new tarball using the ***gzip*** compression alogrithm. While it's not a strict requirement, it's also good practice to add the .tar extension to all tarballs as well as whatever extension is used by the compression algorithm.

```
darkstar:~$ tar -czf /tmp/tarball.tar.gz /tmp/tarball/
```

Reading Documents

Traditionally, UNIX and UNIX-like operating systems are filled with text files that at some point in time the system's users are going to want to read. Naturally, there are plenty of ways of reading these files, and we'll show you the most common ones.

In the early days, if you just wanted to see the contents of a file (any file, whether it was a text file or some binary program) you would use ***cat***(1) to view them. ***cat*** is a very simple program, which takes one or more files, concatenates them (hence the name) and sends them to the standard output, which is usually your terminal screen. This was fine when the file was small and wouldn't scroll off the screen, but inadequate for larger files as it had no built-in way of moving within a document and reading it a paragraph at a time. Today, ***cat*** is still used quite extensively, but predominately in scripts or for joining two or more files into one.

```
darkstar:~$ cat /etc/slackware-version
Slackware 14.0
```

Given the limitations of ***cat*** some very intelligent people sat down and began to work on an application to let them read documents one page at a time. Naturally, such applications began to be known as “pagers”. One of the earliest of these was ***more***(1), named because it would let you see “more” of the file whenever you wanted.

more

more will display the first few lines of a text file until your screen is full, then pause. Once you've read through that screen, you can proceed down one line by pressing ENTER, or an entire screen by pressing SPACE, or by a specified number of lines by typing a number and then the SPACE bar. ***more*** is also capable of searching through a text file for keywords; once you've displayed a file in ***more***, press the / key and enter a keyword. Upon pressing ENTER, the text will scroll until it finds the next match.

This is clearly a big improvement over ***cat***, but still suffers from some annoying flaws; ***more*** is not able to scroll back up through a piped file to allow you to read something you might have missed, the search function does not highlight its results, there is no horizontal scrolling, and so on. Clearly a better solution is possible.

In fact, modern versions of ***more***, such as the one shipped with Slackware, do feature a **back** function via the `b` key. However, the function is only available when opening files directly in ***more***; not when a file is piped to ***more***.

less

In order to address the short-comings of ***more***, a new pager was developed and ironically dubbed ***less***(1). ***less*** is a very powerful pager that supports all of the functions of ***more*** while adding lots of additional features. To begin with, ***less*** allows you to use your arrow keys to control movement within the document.

Due to its popularity, many Linux distributions have begun to exclude ***more*** in favor of ***less***. Slackware includes both. Moreover, Slackware also includes a handy little pre-processor for ***less*** called `lesspipe.sh`. This allows a user to execute ***less*** on a number of non-text files. `lesspipe.sh` will generate text output from running a command on these files, and display it in ***less***.

Less provides nearly as much functionality as one might expect from a text editor without actually being a text editor. Movement line-by-line can be done ***vi***-style with `j` and `k`, or with the arrow keys, or `ENTER`. In the event that a file is too wide to fit on one screen, you can even scroll horizontally with the left and right arrow keys. The `g` key takes you to the top of the file, while `G` takes you to the end.

Searching is done as with ***more***, by typing the / key and then your search string, but notice how the search results are highlighted for you, and typing `n` will take you to the next occurrence of the result while `N` takes you to the previous occurrence.

Also as with ***more***, files may be opened directly in ***less*** or piped to it:

```
darkstar:~$  less
/usr/doc/less:/README
darkstar:~$  cat
/usr/doc/less:/README
/usr/doc/util-linux:/README | less
```

There is much more to ***less***; from within the application, type `h` for a full list of commands.

Linking

Links are a method of referring to one file by more than one name. By using the ***ln***(1) application, a user can reference one file with more than one name. The two files are not carbon-copies of one another, but rather are the exact same file, just with a different name. To remove the file entirely, all of its names must be deleted. (This is actually the result of the way that ***rm*** and other tools like it work. Rather than remove the contents of the file, they simply remove the reference to the file, freeing that space to be re-used. ***ln*** will create a second reference or “link” to that file.)

```
darkstar:~$ ln /etc/slackware-version foo
darkstar:~$ cat foo
Slackware 14.0
darkstar:~$ ls -l /etc/slackware-version foo
-rw-r--r-- 1 root root 17 2007-06-10 02:23 /etc/slackware-version
-rw-r--r-- 1 root root 17 2007-06-10 02:23 foo
```

Another type of link exists, the symlink. Symlinks, rather than being another reference to the same file, are actually a special kind of file in their own right. These symlinks point to another file or directory. The primary advantage of symlinks is that they can refer to directories as well as files, and they can span multiple filesystems. These are created with the *-s* argument.

```
darkstar:~$ ln -s /etc/slackware-version foo
darkstar:~$ cat foo
Slackware 140
darkstar:~$ ls -l /etc/slackware-version foo
-rw-r--r-- 1 root root 17 2007-06-10 02:23 /etc/slackware-version
lrwxrwxrwx 1 root root 22 2008-01-25 04:16 foo -> /etc/slackware-version
```

When using symlinks, remember that if the original file is deleted, your symlink is useless; it simply points at a file that doesn't exist anymore.

Chapter Navigation

Previous Chapter: Booting

Next Chapter: The Bourne Again Shell

Sources

- Original source: http://www.slackbook.org/beta
- Originally written by Alan Hicks, Chris Lumens, David Cantrell, Logan Johnson

slackbook, shell, archive, filesystem