Chapter 3

The Filesystem

If your going to be a good system administrator, you need to know something about the filesystem. If you are coming from a Windows system, you will notice several significant differences in the way the filesystem is laid out. While this may be confusing at first, you will find that these conventions will make administering your system easier.

3.1 File Types

One of the unique things about the UNIX file system is that everything is represented as a file. Files can be divided into three major categories: **regular files**, **directory files**, and **device files** (there are actually several other types of special files, some of which we will talk about later).

3.1.1 Regular Files

A **regular file** is the category most files fall under. They are plain files that contain text or binary data and are stored on a physical device (like a cd, hard drive, usb key, etc.).

3.1.2 Directory Files

A **directory file** is a special file used to organize files. A directory does not contain regular data like regular files, but instead contains information about the files in that directory. Basically, it contains a list of files (of any type) along with the file's **inode** number. **Inodes** will be discussed later, but for now, think of them as a unique identification number for files in the file system.

3.1.3 Device Files

A **device file** is a special file that is used to interact with physical devices attached to the system. They aren't physically stored anywhere and exist only

in memory. When you write to or read from a device file, you are interacting with a physical device. For example, the file `/dev/hda1` is the device that represents your first hard drive partition. There are some device files (like `/dev/null`) that don't actually correspond to physical devices. These are called **pseudo-devices**.

3.2 File Names

Unlike Windows, there are very little restrictions placed on what you name your files. The name must be less than 255 characters, but other than that, you are free to name it what you want. That being said, there are several things to keep in mind.

3.2.1 File Extensions

File extensions are not required in UNIX. You can name a text file `data.jpg` if you really wanted to, but you really shouldn't. The file extensions make it easier for you to tell whats in the file. If you are not sure about a file type or want to confirm, you can use the **file** command. **file** performs tests on the file data instead of checking the file name. For example:

```
aoaks@tsunami:~$ cp hw2.pdf hw2.txt
aoaks@tsunami:~$ cp hw2.pdf hw2
aoaks@tsunami:~$ file hw2*
hw2:     PDF document, version 1.4
hw2.pdf: PDF document, version 1.4
hw2.txt: PDF document, version 1.4
```

Notice that even though the file name has been changed, and even dropped, **file** still determines the actual file type correctly.

3.2.2 Special Characters

You can put whatever you want in your filenames, even shell special characters (with the exception of the '/' character and '\ 0' the **null** character). While you are technically allowed to, you should really try not to use these characters. Every time you need to use the file, you will need to escape all of the special characters in the name (this means spaces too, which is why, as you will see, none of the system directories have spaces in them). You should definitely never use the '-' (hyphen) at the beginning of a filename. When you type it in, it will be interpreted as an option to whatever command you are using. The '-' is not a shell special character, so you can't simply escape it either. The best way to deal with it is to not do it.

3.2.3 Hidden Files

The **ls** command will show you the contents of the current directory. However, just like in other operating systems, there are files that you don't always want displayed (for example, configuration files). In UNIX, to make a file hidden, start its filename with a '.' (period). To show hidden files, use **ls** with the '-a' or '-A' option (if you are curious about the difference between them, read the man page). For example:

```
aoaks@tsunami:~/wallpaper$ ls
niceview.jpg
goldengatebridge.jpg
aoaks@tsunami:~/wallpaper$ mv niceview.jpg .niceview.jpg
aoaks@tsunami:~/wallpaper$ ls
goldengatebridge.jpg
aoaks@tsunami:~/wallpaper$ ls -A
.niceview.jpg
goldengatebridge.jpg
```

3.2.4 Case Sensitivity

File names in UNIX are case sensitive. Therefore, you can have several files in the same directory with the same name, so long as their capitalizations are different. For example the following is perfectly legal:

```
aoaks@tsunami:~/case$ ls
data  Data  DATA
```

While such things are allowed, they are general avoided because it makes things unnecessarily confusing.

3.3 Filesystem Structure

Like most other operating systems, UNIX uses a hierarchical directory structure. Each mounted filesystem has a **root** (top level) directory which contains files and subdirectories. What throws most people coming from an operating system like windows is the way the filesystems are mounted. On Windows, each filesystem mounted on the system is assigned its own drive letter (like `C:`, `D:`, etc) and the file system is mounted under it. In this system, the root directory of each partition is accessed through the different drive letters. For example `C:\` would be the root directory on the main filesystem, `D:\` would be the root directory on the second file system, etc.

UNIX handles mounted in a different fashion. In UNIX, everything is accessed under a single filesystem, and individual file systems are mounted at **mount points**. A **mount point** is the directory in the main file system where you attach the root directory of another filesystem. The main filesystem is

mounted on the root directory and is appropriately named the **root filesystem**. All other file systems are mounted on directories under the root directory. For example, `/dev`, `/proc`, and `/tmp` are actually each separate filesystems. At boot, their root directories are mounted at their respective mount points.

3.4 Directory Structure

The UNIX directory structure is set up so that similar data is stored together. Let's look at some of the directories in the root directory, looking at what the directory names mean and what they contain. These folders are based on a convention in Linux called the **Filesystem Hierarchy Standard**.

/bin (Binaries) This is where main program binaries are located. Most of the commands without specifying a path are located in a `bin` directory.

/sbin (System Binaries) This is where programs used for maintenance or administrative tasks are stored. These are not usually used by anybody other than the system administrator, both because their use is limited to administrative task and because many of them are only usable as the system administrator.

/home (Home Directories) This is where each user's home directory is stored. Each user gets a home directory in which to store their personal data and settings information.

/usr (User Shareable) This is a secondary hierarchy for user shareable data. Sort of like a secondary root directory. It contains many similar subdirectories as '/', though the contents are usually not as essential to system operations (in particular, are not required for the system to boot).

/dev (Devices) This is where all of the special device files are located. This contains the links to all of the devices (e.g. hard drives, cds, terminals, etc.) connected to your system.

/var (Variable Data), This is where all of the 'variable' data on the system is stored. This includes things like mail files, print jobs, and system logs.

/tmp (Temporary Data) This directory contains temporary space which can be used by all users on the system. This directory is cleaned out regularly by the system administrator, so don't leave important things here.

/boot (Boot) This is where information and data used for the system boot process is stored. It includes things like kernel images and boot loader configuration.

/proc (Process) This is a virtual file system (it takes up no disk space) that contains information about system hardware and any currently running processes.

/etc (Et cetera) This is where all of the system configuration files are stored. These files are usually all plain text and contain no binary data.

3.5 Navigation

Getting around the file system is accomplished by using the **cd** (change directory) command. **cd** takes on argument: the directory you want to change to. If you use **cd** without an argument, it will take you to your home directory (the value of `$HOME`). To find your current directory, use the **pwd** (print working directory) command.

3.5.1 Absolute vs. Relative Path

When you specify a file, you must specify it relative to some path. The **absolute path** is the path to the file with respect to the **root** directory. The **relative path** is the path to the file relative to the current directory. For example, if my current directory is `/foo/bar/baz/` and I want to specify the file `results.dat` in the `data/` directory, the absolute path would be `/foo/bar/baz/data/results.dat`, while the relative path would be `data/results.dat`.

If you want to specify the absolute path to a file, the path must begin with a '/'. This tells the shell to begin looking in the root directory. If the path you specify doesn't start with a '/', the shell will assume you are giving a relative path.

3.5.2 Special Directories: . and ..

Every directory in the filesystem contains two special directories: the '.' and '..' directories. These directories reference the current directory and the parent directory respectively. This provides a way for you to reference the current directory and parent directory without having to specify their absolute paths. For example, if you wanted to restore a backup copy of your thesis from your backup directory, you might do something like:

```
cp backup/thesis.tex .
```

The '..' directory provides a convenient way of moving around the directory structure in addition to making file references easier. For example, if your current directory is `/opt/really-local/packackes/samba-2.2.12/bin` and you wanted to move to `/opt/really-local/packackes/samba-2.2.12/etc`, rather than doing:

```
cd /opt/really-local/packages/samba-2.2.12/etc
```

you could do something like:

```
cd ../etc
```

(For those who were curious about the difference between `ls -a` and `ls -A` but were too lazy to look it up, using '-a' shows you *all* hidden files, whereas using '-A' shows you *almost all* hidden files, skipping '.' and '..')

3.5.3 The $PATH

After hearing that all files are specified relative to some path, the very perceptive among you may have thought "wait... if the commands I type in are regular files like everything else, why don't I have to specify a path to them?" The answer is: you are. If you want to explicitly specify the path to a command, you can do so by using an absolute path, like:

```
aoaks@tsunami:~$ /bin/cat hello.txt
Hello World
```

or a relative path, like:

```
aoaks@tsunami:/bin$ pwd
/bin
aoaks@tsunami:/bin$ ./cat hello.txt
Hello World
```

So what's happening when you don't specify a path, like:

```
aoaks@tsunami:~$ cat hello.txt
Hello World
```

What command it that running? Well, when you specify a command without giving any sort of path, the shell will try to find the command in the paths stored in your `$PATH` environment variable. A typical `$PATH` variable looks like:

```
aoaks@tsunami:~$ echo $PATH
/usr/local/bin:/usr/local/sbin:/bin:/usr/bin:/usr/sbin:
/opt/mlocal/bin:/usr/bin/X11
```

As you can see, most of them are `bin` or `sbin` directories, delimited by a ':' (colon). When the shell gets a request to look up a command in the $PATH, it tries each of these directories in sequence until it finds one that contains the desired command. Its important to note that the shell will run the first match it finds, so if there is a `/usr/local/bin/cat` and a `/bin/echo`, the shell will run the one in `/usr/local/bin`. If for some reason you need to run a different `cat`, you will need to explicitly specify the path to that `cat`. To find out where exactly the program you would run is located, use the **which** command:

```
aoaks@tsunami:~$ which cat
/bin/cat
```