Chapter 1

A Guided Tour of the Command Line

When learning something new, the first step is usually the hardest. You need to take the plunge and dive right into a new way of thinking. Working with the command line is like learning a new language—you have to start with simple tasks and take small steps before you can move ahead to the real nitty-gritty.

When you visit a city you have never been to, it can be useful to take a guided tour. Your tour guide can show you the main landmarks and give you basic explanations of what you see. This is what I'm going to do in this chapter. Think of it as a bus tour of a foreign city, where you'll see the sites but you won't get off and wander on your own.

If you have never used the command line, this chapter gives you a brief introduction to entering commands and using Terminal, the Mac OS X program you use to run these commands. This tutorial introduces a few basic commands, and shows you what happens when you run those commands, but I don't give thorough explanations of any of the commands used here. Don't worry; I'll go into detail about all of them in later chapters. The goal of this chapter is to walk you through a few commands and show you that using the command line is not really as difficult as you may have thought. To get the most out of this chapter, sit down in front of your Mac and follow me as I take you on a guided tour of the command line.

NOTE *If you are familiar with the command line, you can skip this chapter; you probably know everything that's presented here.*

Opening Terminal

Terminal is the program Apple includes in Mac OS X to provide the interface between the commands you type and the operating system. Terminal itself doesn't do much—it merely passes commands and data on to a shell (another program that interprets these commands) and displays the results of these commands.

Start by opening Terminal (see Figure 1.1). This program is located in the `Utilities` folder of your `Applications` folder (or, to use the Unix convention, `/Applications/Utilities`). Double-click the Terminal icon.

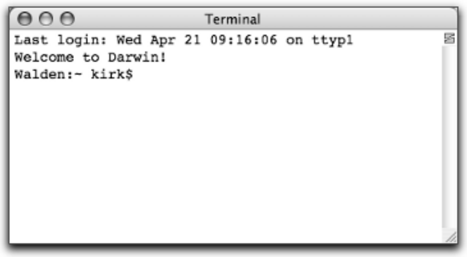
The Terminal window displays, showing something like Figure 1.2.

The text in this window tells you several things:

- The first line shows the date and time of the last login, followed by the terminal device ("ttyp1") being used.
- The second line is the Message of the Day. The default message of the day for Mac OS X (as of version 10.3) is `Welcome to Darwin!`, Darwin being the name of the Mac implementation of BSD Unix.
- The third line is the *prompt*. It contains several parts:
  - It first shows the name of the computer being used—in my case, `Walden`. This name comes from the Sharing panel in the System Preferences.
  - The current directory or folder is shown after the colon (`:`) following the computer name. When you open a new Terminal window, this is by default your home folder, represented by the ~ shortcut.
  - The next part of the prompt is the name of the user, `kirk`, who is logged in. This is the short user name, not the user's full name. (Obviously, your computer name and user are different than mine, which are shown in this example.)
  - The final part of the prompt is the actual prompt character, or `$`. If you're using the `tcsh` shell, you'll see `%` here instead.

FIGURE 1.1
The Terminal icon in the Utilities folder of your hard disk

FIGURE 1.2
A new Terminal window



```
Last login: Wed Apr 21 09:16:06 on ttyp1
Welcome to Darwin!
Walden:~ kirk$
```

The entire line containing

```
Walden:~ kirk$
```

is known as the *prompt*; it indicates that you can type commands. If Terminal is working on a command or displaying certain processes, you don't see a prompt. In that case, you can always open a new Terminal window to type commands; you can open an unlimited number of Terminal windows, called sessions, at any time.

Typing Your First Command

Now that we've gotten through the basics, you're ready to type your first command. Let's start with `echo`, a simple command that displays what follows the command name in the Terminal window.

(Note: the commands and text you are to type are shown following the prompt character, `$`, but without the computer name and user name. You don't type the prompt text; just type what follows the `$` character.)

Type the following:

```
$ echo Hello
```

then press Enter or Return.

Terminal displays each character as you type it. The `echo` command writes arguments to the standard output—in this case, Terminal. After you press Return, it displays the following line:

```
Hello
```

then displays the prompt again, showing that Terminal is ready to go on to its next task, as in Figure 1.3.

Your Mac just said hello to you! Now, you can go even further—after all, you don't know who it was saying hello to. Try this command:

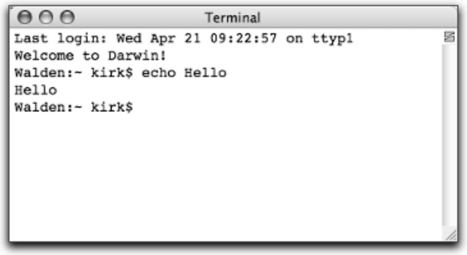
```
$ echo Hello $USER
```

My Mac says:

```
Hello kirk
```

What does yours say? It should say `Hello [`*`your user name`*`].`

FIGURE 1.3
A Terminal window showing the `echo Hello` command and its results



```
Last login: Wed Apr 21 09:22:57 on ttyp1
Welcome to Darwin!
Walden:~ kirk$ echo Hello
Hello
Walden:~ kirk$
```

That was easy, wasn't it? You've just run your first command with the Mac OS X Terminal. This was certainly a simple command, but you have seen how Terminal works, and you have used basic command syntax.

All commands use a specific syntax; for simple commands, this is often just

```
command argument
```

The command is the order you give to your computer, and the argument is what the order acts on. Arguments are required for some commands; other commands run with no arguments. In the line of text you typed above, the command was `echo` and the argument was the text you wanted Terminal to display. (Note that when presenting commands in this book, they are shown in `monospace` font, and arguments follow in the same typeface. This is a standard typographical convention for presenting commands.)

The syntax for the `echo` command is as follows:

```
echo string...
```

This means that to run the command, you must type `echo`, then a string of text. This command writes arguments (the text string) to the standard output, whether it be Terminal, as in the above example, or any other output specified, such as a file (see below). You can now try typing the `echo` command with other text if you want.

Reading Directory Contents

Let's explore some other things you can do with Terminal. In the following series of commands, you will

1. Create a new directory (the Unix word for folder).
2. Examine the contents of the directory.
3. Create a new file.
4. Tell your computer to write some text to that file.
5. Read the file.
6. Delete the file and directory.

The Terminal prompt shows that you are in your home (~) directory. Each user has a home directory that contains their personal files. In the Finder, you can go to this directory by clicking the Home icon (Figure 1.4) in a Finder window sidebar.

FIGURE 1.4
The home folder icon in a Finder window sidebar

Let's see what's in this directory. If you recall from looking at it in Finder windows, it contains a few folders. Type the following (remember, only type what is *after* the prompt):

```
$ ls
```

(This is the letter l, not the digit 1.) The ls command lists the contents of a directory. Terminal displays the following:

```
Desktop    Library  Music     Public
Documents  Movies   Pictures  Sites
```

This is a list of everything in your home folder. (You may see a different list if you have added files or folders to your home folder.)

Unlike when viewing files and folders in the Finder, this list doesn't tell you which of the above items are files or folders. You can find out by typing the following:

```
$ ls -F
```

Terminal displays this list:

```
Desktop/    Library/  Music/     Public/
Documents/  Movies/   Pictures/  Sites/
```

In the above example, `-F` is an *option* for the `ls` command; it is also case-sensitive: `-F` is not the same as `-f`. Options tell certain commands to do things in a slightly different way. This option tells Terminal to display a slash (/) immediately after each pathname that is a directory, an asterisk (*) after each executable (application), etc. The slashes here show us that these are directories. If any of the above items were files, there would be nothing after their names.

Creating a New Directory

Now you're going to create a new directory called `Test`. Type the following:

```
$ mkdir Test
```

The `mkdir` command, as its name suggests, makes new directories. Let's check to make sure that this directory has been created by repeating the `ls -F` command:

```
$ ls -F

Desktop/    Library/  Music/     Public/  Test/
Documents/  Movies/   Pictures/  Sites/
```

There it is: Test/, the directory that you just created.

Now we are going to move into that directory, using the `cd` command:

```
$ cd Test
```

The `cd` command changes the current working directory; this is similar to double-clicking a folder in the Finder. After running this command, the prompt changes to show that we are now in the `Test` directory:

```
Walden:~/Test kirk$
```

This is the part after the colon showing `~/Test`. As we have already seen, ~ is a shortcut for your home directory, and the slash means that the following directory is inside the home directory.

Creating a New File

Let's now create a new, empty file inside the `Test` directory. Type this:

```
$ touch testfile
```

The `touch` command is typically used to update file access and modification times, but it can also create a new file; the argument, `testfile`, is the name we're giving to the file.

Let's check to make sure the file was created. Type the following:

```
$ ls -F
```

which should display:

```
testfile
```

Remember that the `-F` option for the `ls` command shows a `/` following a directory; it shows nothing for files. So we now have a new, empty, file called `testfile` sitting in the `Test` directory, just waiting for something to do.

Writing Text to a File

Since this file is doing nothing, you might as well write something to it. How about writing `Hello [username]` in this file? To do so, you can use the `echo` command that you learned above. Type the following:

```
$ echo Hello $USER > testfile
```

This tells Terminal to `echo` the text `Hello [username]` to the file called `testfile`. Let's check and make sure it worked. Several commands are available to display the contents of your files; one of them is `cat`. Type this:

```
$ cat testfile
```

Terminal should display:

```
Hello [your user name]
```

But since we only see this in the Terminal window, we don't have the same impression as when we open a window in an application. Let's see how the text looks in this file. Type:

```
$ open .
```

(Make sure you type `open`, then a space, then a period.)

This tells the Finder to open the current directory (the `.` is a shortcut for that) in a new window. You should see a new Finder window, entitled Test, with a file, called `testfile`, inside it.

Double-click this file, which should open with TextEdit (a simple text editor that comes with Mac OS X) and display the text `Hello [username]`, as in Figure 1.5.

Quit TextEdit to close the file, then switch back to Terminal by clicking its icon in the Dock.

Deleting Files and Directories

Now that we have finished our brief demonstration, we need to clean up a bit. We don't really need to keep that file and folder, so let's delete them.

Warning *Using the command line can be risky. Unlike working in the Finder, some tasks you carry out on the command line are absolute and cannot be undone. The command I am about to present,* `rm`*, is very powerful. It removes files permanently and completely. There is no getting them back after running this command, so use it with great care, and always use the* `-i` *option, as explained below, so Terminal asks you to confirm deleting each file.*

Your prompt should now look something like this, showing that you are still inside the `Test` directory you created earlier:

```
Walden:~/Test kirk$
```

Type the following:

```
$ rm -i testfile
```

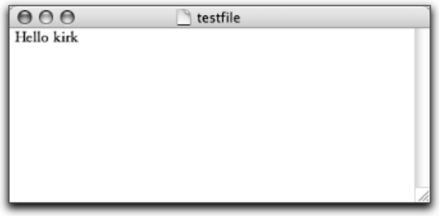
The `rm` command removes files and directories. In the above example, it removes the file called `testfile`. Be careful with this command; it doesn't just put files in the trash, it removes them completely. The `-i` option tells Terminal to run the `rm` command in interactive mode, asking you to make sure you want to delete the file. Terminal asks:

```
remove testfile?
```

Type `y`, for yes, then press Return or Enter, and the file is removed. If you wanted to leave it there, you could just type anything other than `y` or press Return.

We should check to make sure the file is gone:

```
$ ls
```



Hello kirk

Figure 1.5
The `Hello [username]` text as displayed by TextEdit

After typing `ls`, you should just see a prompt. Terminal doesn't tell you that the directory is empty, but it shows what's in it: nothing.

Now, move up into your home folder. Type:

```
$ cd ..
```

This is the same command we used earlier to change directories. Here the command tells Terminal to go up in the directory hierarchy to the next directory (the `..` is a shortcut for the parent directory)—in this case, your home directory.

Type `ls` again to see what's in this directory:

```
$ ls
```

You should see something like this:

```
Desktop    Library  Music     Public  Test
Documents  Movies   Pictures  Sites
```

The `Test` directory is still there. It's easy to delete this directory using `rm`. Type the following:

```
$ rm -d -i Test
```

The `-d` option tells this command to remove directories.

When Terminal displays:

```
remove Test?
```

Type **`y`**, then press Return or Enter. (If you didn't remove `testfile`, as explained above, the `rm` command cannot delete the directory. It will not delete directories that are not empty.)

Make one final check to see if the directory has been deleted.

```
$ ls

Desktop    Library  Music     Public
Documents  Movies   Pictures  Sites
```

Summing Up

If you worked through this brief demonstration, you successfully typed commands in Terminal using the Unix command line. You created a directory (folder), created a file, wrote text to it, then deleted the file and the directory. And all that with some very simple commands. While you have not accomplished anything extraordinary, you have seen that using Terminal is not really that complicated. All it requires is a bit of time to learn the different commands and their arguments and options. If you move ahead slowly, learning as you go on, rather than trying to memorize dozens of commands, you'll soon find that you are not only comfortable with the command line, but that you can do things that help you save time and give you much more power than when working with windows and icons.