A Brief Introduction to the Command Line

Hautahi Kingi

Introduction

A **shell** is a computer program like any other. But its primary purpose is to read commands and run other programs, rather than to perform tasks/calculations itself.

We interact with the shell through a command-line interface or **terminal**. We do so by entering commands as text input. The terminal reads the text input, interprets the commands and sends instructions to the shell, which then executes the appropriate operating system functions. Once the commands are carried out, the shell communicates this with the terminal which then prints its output.

Getting Started

Type *whoami* into the terminal.

```
$ whoami
hautahikingi
$
```

The command's output is the ID of the current user. When we type *whoami*, the shell finds a program called whoami, runs that program, displays that program's output (your username), then displays a new prompt to tell us that it is ready for more commands.

Now let's try the following.

```
$ pwd
/Users/hautahikingi
```

This stands for *print working directory*. When you first login, your current working directory is your home directory.

Exploring a Directory

To find out what is in your home directory, type

```
$ ls
Desktop Dropbox Pictures Documents essay.txt Downloads Movies
```

ls prints the names of the files and directories in the current directory in alphabetical order, arranged neatly into columns.

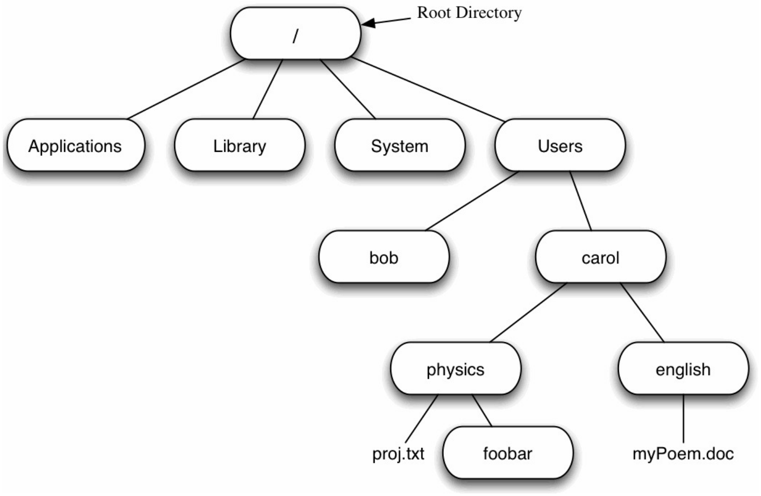
A number of commands have extra options or features which we can call through the use of a **flag**.

```
$: ls -l
total 8
drwx------+  53 hautahikingi  staff   1802 Jul 21 22:34 Desktop
drwxr-xr-x+  21 hautahikingi  staff    714 Jul  8 13:25 Documents
drwx------+ 503 hautahikingi  staff  17102 Aug  5 11:07 Downloads
\vdots
```

The -l flag on the *ls* command is short for long format and it displays a more detailed listing of the files within the current working directory, including file size and date last modified. Note that there is a space between ls and -l: without it, the shell thinks we're trying to run a command called ls-l, which doesn't exist.

Try the -F flag. What does this do?

File Hierarchy



The home directory of *carol* contains two sub-directories - *physics* and *english*. The full path to the file *myPoem.doc* is */Users/carol/english/myPoem.doc*. Notice the two meanings for the / character.

File and Directory Navigation

We can use *cd* followed by a directory name to change our working directory. *cd* stands for *change directory*, which changes the shell's idea of what directory we are in. In the example above, we can type

```
$ cd /users/carol/physics
```

cd doesn't print anything, but if we run *pwd* after it, we can confirm that the current working directory is *physics*.

```
$ pwd
/users/carol/physics
```

If we run *ls* without parameters now, it lists the contents of /users/carol/physics

```
$ ls
proj.txt  foobar
```

File and Directory Navigation

We do not have to type the full path to the directory. Instead, we can use **relative path references**. When *cd* is followed by a folder path that does not begin with the root directory (/), it assumes that you are first referencing the current working directory. In the above example, because we were located in */users/carol/* we only needed to type

```
$ cd physics
```

We also do not have to type the entire name of a directory. *Tab Completion* is very handy in situations where the filename is long. Pressing tab asks the terminal to guess what you are trying to type. For example, if we are located in /Users/carol/ and type

```
$ cd ph
```

and then press tab, the shell automatically completes the *physics* directory name for us. If, however, there was another directory called *philosophy*, the user would need to type *cd phy* before pressing tab.

Creating Directories

Let's now make a subdirectory called *my_thesis* in the Dropbox folder.

```
$ cd Dropbox
$ mkdir my_thesis
$ cd my_thesis
$ pwd
/Users/hautahikingi/Dropbox/my_thesis
```

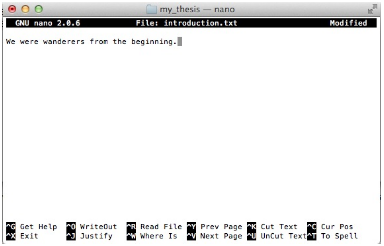
Here, I navigated to the Dropbox folder, made the *my_thesis* directory, and then navigated into the newly created directory.

Creating files with a text editor

Let's now create a new file called *introduction.txt* using a text editor.

```
$ nano introduction.txt
```

I am using **Nano** here because it is the default text editor on a Mac and it is very easy to use. The equivalent on a Windows machine would be **Notepad**. Let's type in a few lines of text.



```
my_thesis — nano
  GNU nano 2.0.6              File: introduction.txt                    Modified

We were wanderers from the beginning.
```

In nano, use Control-X to quit the editor and return to the shell. Make sure you save in the process. We can check the changes we've made to a particular file by using the *cat* command.

```
$ cat introduction.txt
We were wanderers from the beginning.
```

Command Line/GUI perform the same tasks

You obviously don't need to create files using the command line. You can do it all the usual way as well. Imagine I just typed up some matlab code which I saved into the my_thesis folder as *code.m*. Now lets look at:

```
$: ls
code.m introduction.txt
```