# Getting Started with Mac OSX Command Line

## *1. The OSX Shell (bash)*

When you log into OSX, you are immediately placed into a command line interpreter called a shell. There are many shell programs available for OSX but the default shell in OSX is called bash. In a shell you may enter commands where you see the command prompt, like so:

```
dt228@machost:~$
```

The prompt above conveys certain information about the server and environment as follows:

| | |
|---|---|
| machost | The name of the user who is logged in |
| @ | Means "on" or "at" |
| machost | The system's host name |
| : | Separator (could be anything) |
| ~ | The current directory (or folder if you prefer). ~ is an alias for "/home/" |
| $ | Conventionally, the character denoting a command prompt |

Run the following commands and the command prompt in the following sequence. When you do this there will be certain output displayed after each, not shown here.

**NOTE 1:** When typing commands into a shell, you must type them exactly as shown. Spaces are significant. Commands are case-sensitive. All punctuation is significant.

**NOTE 2:** Do not type the "`dt228@machost:~$`" in any of the following exercises. This is shown only as a guide here.

```
dt228@machost:~$ echo "hello from User OSX"

dt228@machost:~$ pwd

dt228@machost:~$ mkdir lab1

dt228@machost:~$ ls

dt228@machost:~$ cd lab1

dt228@machost:/lab1$ pwd

dt228@machost:/lab1~$ cd

dt228@machost:~$ history
```

What do you think the previous commands did? The OSX help command is called *man* (manual).

```
dt228@machost:~$ man man

dt228@machost:~$ man echo

dt228@machost:~$ man mkdir

dt228@machost:~$ man ls
```

Note the format of a man page. Also, note the conventions used for command line arguments and the use of the dash (-).

## *2. Command line navigation and editing*

In the like event that you make a mistake when typing in commands you need to know how best to correct them. Typing everything in again is an option but definitely not the most efficient. Commit the following to memory to save you time when on the command line

| Key sequence | Full name | What it does |
|---|---|---|
| ← | Left Arrow | Move left one character if not already fully left |
| → | Right Arrow | Move right one character if not already fully right |
| ↑ | Up Arrow | Recall the previous command typed |
| ↓ | Down Arrow | Go to the next command typed |
| Ctrl-e | Control-a | Go to the start of the command line |
| Ctrl-e | Control-e | Go to the end of the command line |
| Ctrl-r | Control-r | Start a search for a previously entered command. Followed by typing one or more characters from that command and pressing ENTER when selected |
| Ctrl-l | Control-l | Formfeed. Clear the screen |
| Tab | File name completion | Pressing the TAB key when tying a file name may complete the typing of it for you if bash can guess what you intent |

## *3. The OSX host characteristics*

Now, let's explore the server machine itself. Run the following commands. When you do this there will be certain output displayed after each, not shown here

```
dt228@machost:~$ cd

dt228@machost:~$ hostname

dt228@machost:~$ uname -a

dt228@machost:~$ df -h

dt228@machost:~$ ps aux | more
```

The last command is an example of a pipeline. There are two commands being issued at the same time. The first one's output is "piped" to a screen pager which allows you to see all of the output of a command which would otherwise exceed the number of visible lines on your screen.

What did each of the previous commands do?

Now, let's look a little more at the user environment.

```
dt228@machost:~$ echo $PATH
```

The variable PATH is called an environment variable. It's value is available to you in your shell commands and to all commands (processes) executed in your shell. PATH contains a colon separated list of system directories that are searched, in the order given, for commands you type in. That brings up an important learning point for this lab. All commands you type are actually implemented as files somewhere in the OSX file system. To find out where a file for a particular command is location you can use the *which* command

```
dt228@machost:~$ which echo

dt228@machost:~$ which ls
```

You can create and process your own variables like too. Notice that when reading back the value of a shell variable you prefix it with the $ symbol but not when setting its value. You may not put spaces between the variable, the operator and its value during assignment.

```
dt228@machost:~$ a=1

dt228@machost:~$ echo $a

dt228@machost:~$ b=1

dt228@machost:~$ expr $a + $b

dt228@machost:~$ unset a

dt228@machost:~$ echo $a

dt228@machost:~$ expr $a + $b
```

You can capture the output of commands into a variable too using the backticks or $().

```
dt228@machost:~$ c=`expr 1 + 1`

dt228@machost:~$ echo $c
```

```
dt228@machost:~$ d=$(echo "This is a captured string")

dt228@machost:~$ echo $d
```

## *4. Creating and processing file content on the command line*

```
dt228@machost:~$ echo "This is the first line in the file" > myfile

dt228@machost:~$ echo "This is the second line" >> myfile
```

Note that '>' redirects output from a command to the named file, creating it or overwriting it. By contrast '>>' append content to an existing file or creates it if doesn't exist

```
dt228@machost:~$ ls -l /

dt228@machost:~$ ls -l / > rootfiles

dt228@machost:~$ cat rootfiles
```

A very important file processing command is 'cat' (concatenate), which means to join together

```
dt228@machost:~$ echo "file 1" > file1

dt228@machost:~$ echo "file 2" > file2

dt228@machost:~$ cat file1 file2

dt228@machost:~$ cat file1 file2 > file3

dt228@machost:~$ ls file*

dt228@machost:~$ cat file3
```

You can create short file with cat as follows

```
dt228@machost:~$ cat > sometext

Enter some lines of text here

^D
```

The last line means you type a Ctrl-D on the last line with nothing else before or after

```
dt228@machost:~$ cat sometext
```

```
```

## 5. More command pipelining

```
dt228@machost:~$ echo '2 + 4 + 6 * 8 + 1 * 3' | bc

dt228@machost:~$ ls | sort -r

dt228@machost:~$ ps -ef | grep root

dt228@machost:~$ ps -ef | grep root | sort -r
```

## 6. Creating and editing files with vi

The default text editor on UNIX (and OSX) systems is called vi (pronounced vee-eye). vi is a dual mode edit which can take some getting used to. Once proficient, users tend to be more productive than in conventional editors.

- Scan through the introduction to vi here: http://www.infobound.com/vi.html
- Use the manual
- Use vi's own internal help system

## 7. Creating and editing files with nono

An alternative  text editor on OSX is called TextEdit which you may prefer instead of vi above.

## 8. Writing shell scripts

Using the vi or TextEdit editor you could practice writing shell scripts to exercise the advanced features of the shell language

```
dt228@machost:~$ man bash
```

You can run your scripts using the bash command as follows.

```
dt228@machost:~$ bash thenameofyourscript
```

Create and run the following files or whatever you like. Try to predict what each does before you run them

```
#!/usr/bin/env bash

for n in 2 4 6 8 1 3 5 7
do
    echo $n
```

```
done | sort
exit 0
```

```
#!/usr/bin/env bash

total=0
for n in 1 2 3 4 5 6 7 9 10
do
    total=`expr $total + $n`
done
echo $total
exit 0
```