A chicken-and-egg problem: we cover the shell in session 2, but we need a few shell commands for the examples in session 1. So it helps to have a basic picture of the command line ahead of time.

Exercise
Open a terminal window with a Unix shell. If you are using a Mac, the application “Terminal” does this (OSX has BSD, a flavor of Unix, underneath). Windows users will need to login to the HPC cluster via Putty, see here for instructions.

Look for the command prompt and try typing the command as shown below.

Keep this window open - we’ll using it throughout the tutorial. When you’re done, you can close it by typing “exit.”

A command, typed at the Unix command prompt, looks something like this:

```
[soho: leak@soho batch]$ ls -l -h -F *.sh
```

The prompt. You don’t type this, the system prints it. It might look a bit different, but it typically ends with ‘$ ’

Some arguments

The basic grammar rules in bash are:
• Commands, options and arguments are separated by spaces.

$ echo hi there
hi there

echo simply writes its arguments to stdout.

• We’ll emphasize that again:

In Unix, **spaces and capitalization are always important!** If you omit the space between 'ls' and '-l', you will see an error message like:

```
ls-l: command not found
```

• Options mostly begin with a dash, can usually be given in any order, and usually must come before arguments

• The shell performs expansion **before** it launches the command

So in this example, the actual command that gets run is not 'ls -l -h -F *.sh' but something like 'ls -l -h -F file1.sh file2.sh'

---

**Streams and Filters**

Another useful way to think of the shell is as an environment for plumbing. You have **streams** of text - such as the output of commands. And each command is a **filter** - it accepts a stream (the standard input), does something to it, and produces another stream (the standard output). Ands the shell provides a toolkit for connecting streams and filters together. We'll look more at this metaphor later.