The shell, in a nutshell

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 ‘$’

A command

Some options

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:

```bash
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’

**Autocompletion with Tab**

You can save yourself much typing thanks to the “Tab” key - when you hit Tab, the shell will attempt to finish the command (or filename, or environment variable) that you started. If it gets stuck, type the next letter or two and hit “Tab” again.

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). And the shell provides a toolkit for connecting streams and filters together. We’ll look more at this metaphor later.