Learning to speak - using the language of the shell

Tutorial:
Introduction to Linux/Unix and the HPC environment at NYU

Back to the NYU HPC Wiki
Back to Tutorials index
Back to the Introduction
Before we start
Session 1: Everything is somewhere - hosts, networks and filesystems
Session 2: The computer as a nut - using Unix commands
The shell, in a nutshell

A command, typed at the Unix command prompt, looks something like this:
The basic grammar rules in bash are:

- Commands, options and arguments are separated by spaces.

```bash
$ echo hi there
hi there
```

*echo* simply writes its arguments to *stdout*.

- We’ll emphasize that again:

```
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.

Certain characters have special meaning to the shell, especially any kind of quotes, brackets or symbols:

- Some (*, ?, [], {}), *word* trigger *expansion*, in which the special character or word is replaced by a variable or a list of filenames. There is a good tutorial with examples at [linuxcommand.org](https://linuxcommand.org), and we'll run through a few essentials below.

- Others set up *lists* (;, & &), in which a sequence of commands is executed one at a time, and *pipelines* (|) or *redirection* (<, >, >>) in which the stdout of each command is passed to the stdin of the next command.

- Single and double quotes protect their contents from interpretation by the shell, as does a backslash (\) for the character immediately following.

The prompt. You don’t type this, the system prints it. It might look a bit different, but it typically ends with ‘$’.
A note about dialects: In this tutorial we cover a shell named "bash" (the "Bourne again shell" - a modernized version of an earlier shell written by Stephen Bourne). Bash is the default shell on most Linux systems including the NYU HPC clusters, on OSX and in Cygwin for Windows. Other shells you will likely encounter are `sh`, `csh`, `tcsh` and `ksh`. The syntax and usage is similar across the board but not identical, so if you are using a different shell to `bash`, first read its man page!

Exercise
Open a terminal window. As we go through the next few sections, type some of the commands into it. (You might want to first find a directory with a few files in it so the commands have something to report)