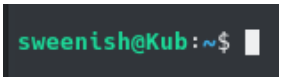
Snippets from The Linux Command Line[1]

Intro to the shell

One objective of this class is to become more familiar with Linux operating systems, specifically, being comfortable using the shell to issue commands to the computer.



Error! Reference source not found. shows a typical shell prompt. This prompt shows the user name at (@) the machine name, followed by the name of the working directory (current folder), and a character that represents the type of privileges allowed. The dollar sign ($) indicates a regular user, where a pound sign (#) indicates superuser privileges.

The solid white block is the cursor. From here, we can issue commands to the computer.

Keyboard shortcuts

While terminal emulators (run from a GUI environment) can incorporate the mouse, it is better to learn the keyboard shortcuts. This is because many servers simply do not have a GUI installed, and other forms of access such as ssh may not allow actions to be completed with a mouse. The table below shows a ^ (carat) symbol, this does NOT mean that we press [SHIFT] + [6], instead it means that we press the [CTRL] key. The carat is used to represent the [CTRL] key in the shell.

Some Keyboard Shortcuts

Key(s)	**Action**
[UP ARROW]	See previous commands in history
[DOWN ARROW]	Move through history toward the present
[TAB]	Filename completion
[^] + [c]	Break, handy for force-quitting programs
[^] + [a]	Move cursor to beginning of line
[^] + [e]	Move cursor to end of line
[ALT] + [f]	Move cursor forward one word
[ALT] + [b]	Move cursor back one word
[LEFT ARROW]	Move cursor back one character
[RIGHT ARROW]	Move cursor forward one character

[1] William Shotts, The Linux Command Line: A Complete Introduction (San Fransciso: No Starch Press, 2012)

Wildcards

Wildcards allow for mulitple selections of similarly named files. For example, to see all source code files in a directory, the command ls -al *.cpp can be issued. The * symbol is a wildcard that can represent ANY characters (note the plural). So, the command ls -al *.cpp will list all files that end with '.cpp', regardless of what comes before.

Wildcards

Wildcard	**Matches**
*	Any characters
?	Any single character
[*characters*]	Any character that is a member of the set *characters*
[!*characters*]	Any character that is not a member of the set *characters*
[[:class:]]	Any character that is a member of the specified class

Common Classes

Class	**Matches**
[:alnum:]	Any alphanumeric character
[:alpha:]	Any alphabetic character
[:digit:]	Any numeral
[:lower:]	Any lowercase letter
[:upper:]	Any uppercase letter

Some Wildcard Examples

Pattern	**Matches**
*	All files
g*	Any file that begin with a lowercase g
b*.txt	Any file beginning with b followed by any characters and ending with .txt
Data???	Any file beginning with Data followed by exactly three characters
[abc]*	Any file beginning with either a, b, or c
BACKUP.[0-9][0-9][0-9]	Any file beginning with BACKUP. followed by exactly three numerals
[[:upper:]]*	Any file beginning with an uppercase letter
[![:digit]]*	Any file not beginning with a numeral
*[[:lower:]123]	Any file ending with a lowercase letter or the numerals 1, 2, or 3

Expansion

Expansion is the mechanism that allows wildcards to work. The short version is that anything we type into a terminal undergoes some processing before it is actually passed on to our programs. Specifically, wildcards use pathname expansion. All this means is the wildcards are expanded relative the path that is specified.

We also have ~ (tilde) expansion. The command 'echo ~' will show you the absolute path to your home directory. This is just a special case of pathname expansion.

There is also arithmetic expansion. It would look something like this: 'echo $((2+2))'. We would see '4' printed to the screen.

There is also brace expansion. Where the wildcards above allow us to select many files/folders based on certain criteria, brace expansion can allow us to create or print many things based on specific criteria. An example of brace expansion looks like this: mkdir hw{01..10}

This command will create ten folders that start with the characters 'hw' and are numbered 01, 02, …, 09, 10 (hw01, hw02, …, hw09, hw10). On other platforms the command may work a little differently; the command can be tweaked accordingly.