Introduction to UNIX Programming

Johnathan W. Hill

Why UNIX?

UNIX is a widely used tool in science and the most popularly used programming environment for neuroimaging analyses. There are two reasons why UNIX is so popular in neuroimaging:

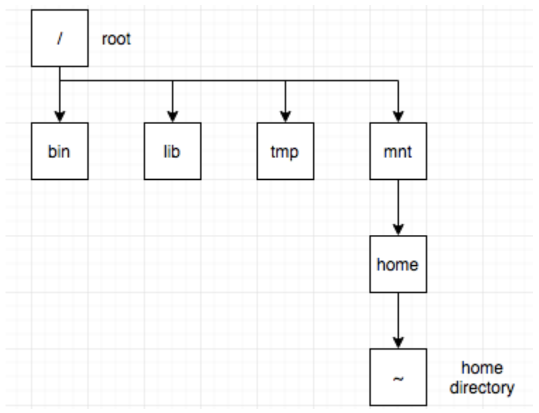
- UNIX is open source, which means it is free! Anyone with access to a computer can essentially use UNIX.
- UNIX is based on a minimalist philosophy, where the "little programs" built into UNIX act as filters. Existing filters are composed with shell scripts to make newer filters. Therefore, project workflow in UNIX can be implemented in a series of steps by writing as little new code as possible.

The Bash Shell

A UNIX shell interprets code sent to the command line. The first UNIX shell was the Thompson shell, or sh. The C shell and Bourne shell followed after. In IBIC, we use the Bourne shell, or bash (Bourne again shell), which is much more similar to the Thompson shell than the C shell. Therefore, commands in sh and bash may be compatible with one another, but will most likely be incompatible with C shell.

Navigating the UNIX Environment

One foundational concept to understand when using UNIX is that information is split into a hierarchy of directories. A relevant metaphor is a tree. All directories originate from the *root* and expand into more complex directory systems the further you deviate from the root. At IBIC, the main organization of this "tree" is split up like this:



The Command Line

A question that may arise when using UNIX for project workflow is, "why not just use the Graphical User Interface (GUI)"? Although GUI's are user friendly and do not take much time to learn, using a GUI for project workflow is much more time-consuming and error-prone than simply using your terminal. For example, if you wanted to move all of the pictures in one of you directories to a new directory, you could easily do so with one line in your terminal. This same task would take a lot longer when just using a GUI and you may accidentally forget to move some of your pictures.

Basic UNIX Commands

Before progressing to more advanced material, understanding these three basic UNIX commands will help you navigate around the UNIX directory structure:

pwd	Prints the path to your current working directory.
ls	Prints a list of the files and directories in your current directory.
cd	Navigates your location in the directory structure.

<u>*Putting it all together*</u>:

1. Type "*pwd*" to check that you are in your home directory. The following path should print to your terminal: "*/mnt/home/<username>*".
2. Otherwise enter the following into your command line: "*cd ~*". The tilde (~) symbol references your home directory.
3. Now, enter "*cd ..*" into the command line. This command will take you a directory higher than your current directory.
4. Enter "*ls*". Here you see a list of home directories. See if you can find your home directory. Try navigating to your home directory with "*cd <username>*".
5. Check that you are in your home directory by typing "*pwd*" into your command line once again. If not, type "*cd ~*" and try again.

Relative vs Absolute Paths*:*

To further understand how to navigate through a directory structure, it is important to distinguish between relative and absolute paths.

<u>Relative paths</u> are "road maps" to locations relative to your original location. For example, let's say I ask you to move two steps backwards, three steps to your left,

then one step forward. You are now two steps backwards, three steps to the left, and one step forward relative to where you originally were.

Let’s work through an example of how you can use relative paths in UNIX.

First of all, go to your current directory. Now, move to root using the following command: “*cd ../../..*”. You are now three steps (directories) back from where you originally began (your home directory).

Absolute paths are much more precise than relative paths. For example, rather than telling you where to move relative to where you already are, I ask you to walk to Tara’s office. It does not matter where you already are, you just walk to Tara’s office.

Similarly to our relative path example, let’s work through another example of how to use absolute paths.

First, change directories to back to root (“*cd /*“). This is a simple example of an absolute path. Now, type “*cd /mnt/home/<username>*”. This is an example of an absolute path to your home directory.

Modifying Behavior Using Flags

Remember how UNIX is built on newer programs built on older programs? Let’s see an example of this principle.

Flags are options in many UNIX commands that allow users to modify the behavior of commands. The term flag comes from the fact that the little hyphen with the letter falling off of it looks like a flag. For example, “*ls –a*” is an extension of ls that allows a user to print all of the files and directories in their current directory. Simply typing “*ls*” will leave out hidden files (files with a “.” in front of them), however, the –a flag allows users to see all files in their directory, including the hidden files.

To see a list of flags than you can use in conjunction with a specific command, you can type “*man <command>*” into your terminal. However, man pages in UNIX can sometimes be very complicated to read. A very useful resource to use in addition to man for looking for flags to use is Google.

OTHER EXAMPLE FLAG OPTIONS IN UNIX

ls -d	Lists the contents of directories.
ls -l	Prints information about file permissions and size.

Standard Input/Standard Output

Distinguishing between standard input and output is critical when using UNIX commands, especially when using more advanced programs. To illustrate the difference between standard input and output, let's run through another example using the command "*cat*".

Cat (short for concatenate) is a command that allows you print the contents of a file and to concatenate new contents into a file.

Type "*cat*" into your terminal and press enter. You will notice that your cursor is no longer in your command line and, instead, is in your terminal. Type "*apple*" into here and press enter. You will notice something very unusual. "*Apple*" is printed back to you. This is because cat accepts the standard input you send to it and it sends it back as standard output. Because you are not specifying where you want the standard output to go, it is printed back to the terminal. Type CNTRL+d (^d) together to exit back to the command line.

A special symbol that you can use to redirect the standard input to cat is a wacca symbol (<>). Type "*cat > testfile*" into your console. Once again, your cursor will move below the command line and prompt you for standard input. Type "*banana*" and ^d once again. What happened here was you specified where the standard output of *cat* is to go. "*Banana*" was sent as standard input into cat, and the standard output of cat was sent as standard input into *testfile* (which was created by sending the output there). Now, type "*cat testfile*" to print the contents of *testfile* to the terminal.

If you type "*cat > testfile*" again and concatenate the input "lemon" into the file, "*banana*" will be replaced by "*lemon*". If you do not want your previous contents to be erased, you can append this extra information back into *testfile* using two waccas (>>). For example: *cat >> testfile*. Read the contents of *testfile* again and check the output.

To further demonstrate standard input/output, the command "*echo*" could be used. Echo is a command that will print its input back to the terminal.. For example, typing "*echo hello world*" will print "*hello world*" to the terminal. Try appending "*hello world*" *to testfile* using the following syntax: "*echo hello world >> testfile*".

PIPES

Piping is another important tool for input/output redirection. To show how pipes are used, let's introduce another basic UNIX command—word count.

Type *wc* into your command line and then type "*hello*" and *^d*. you will see three different numbers printed out behind your input. In this case: 1 1 5.

The first number represents how many lines, the second represents how many words, the third represents how many characters.

You can also use the echo command with the wc command by using a pipe character "|". For example, type "*echo hello world | wc*". What is happening here is the standard output of echo is redirected into wc and the standard output is a count of the lines, words and characters in the phrase "hello world".

Word count also has some very useful flags. For example, "*wc –w*" will provide a count of how many words are in the input sent to it.

NEWLINE CHARACTER:

When pressing "return" to direct to a new line, a newline character is printed out. Although this character is not explicitly printed out to the terminal, *wc* will still count the newline character as a character.

Wild Cards

In order to illustrate another important UNIX concept—wildcards—type "*ls –a .**" into your command line. Here, all of the hidden files in your current directory will be printed out to the console. An asterick (*) can be used to stand in for multiple characters. This is an example of a wildcard.

To better understand wildcards, let's create a few files in our home directory. Type "*touch 2a 2b 3a 3c 3d 2bv*" into your command line. Touch is a command that will automatically make files for you. If you *ls*, you will see that several files have been created in your directory. Now type, "*ls 2**". You will see that all files starting with a "2" have been printed out to your terminal.

Another useful wildcard is "*?*". Typing "*ls 2?*" will print out all of the files that begin with 2 and are followed by a single character (so 2bv is not printed out).

OTHER UNIX CONCEPTS NOT EXPLICITLY COVERED

Other basic commands:

df	Prints out the IP address of your hostname and information about your file systems.

clear	Clears anything on your terminal.
grep	Matches strings of characters in its input.

UNIX CONVENTIONS:

By convention, it is important to use only use lower-case letters and refrain from using upper-case/non-alphabetic/numeric characters for file or directory names (unless truly necessary).