Introduction and a dash of *nix

Overview

1. Command line: what and why?
2. Unix intro
3. Unix command line

Getting started

What is *nix?

- "*nix" refers to a group of operating systems either derived from or inspired by the original AT&T Unix from Bell Labs
  - GNU/Linux is a "Unix-like"
  - macOS is an actual Unix derivative
  - *nix systems follow similar principles and provide similar (software) interfaces
- Unix and its derivatives have entrenched themselves in academia and industry
  - The many tools developed to run on *nix systems are mature and are here to stay
  - General *nix literacy will help you since you have a pretty good likelihood to be developing on a *nix system
- This does not mean that *nix systems are inherently better than other operating systems like Windows
  - Windows also has its own set of tools
  - Some *nix tools have been ported to Windows
  - Windows now has WSL(2) that serves as a Linux living inside Windows

What is a command line?

- The "command line" is a type of interface where *you provide a line of text* that the interpreting software can interpret into commands to perform
  - This interpreting software is known as a "shell"
  - There are also "graphical shells" i.e. the GUIs of Windows and macOS
  - These take an input like a mouse click on a shortcut and interprets it as a command to launch the appropriate application

Why the command line?

- Before we had graphical displays we printers and teletypes (TTYs)
  - `printf()` literally meant to print
- We then moved onto video **terminals**
  - These were a combination display and keyboard, except they could only display text and symbols
  - Nowadays we don't have actual video terminal devices, but we have "virtual terminals" and "terminal emulators" to act like them (e.g. macOS Terminal, iTerm 2, Command Prompt)
- Unix and the many tools for it were developed during these times
- Text serves as a long lasting, reliable interface that is very easy to automate
  - Count the number of GUI changes to Windows, macOS, Android, and iOS over the years
  - How would you automate a GUI?
  - It probably would be more work than writing some commands to be run

Command line basics

- We will focus on the *nix command line shell in this class
- (From now on, when I say "shell" by itself I mean command line shell)
- Shells follows very similar basic syntax no matter what shell (bash, zsh, csh, etc.) you use
- Shells provide you an interface to interact with the system via its directories (folders) and files
  - You can navigate through directories
  - You can modify files
  - You can launch applications
- Most shells feature some sort of *tab completion*, where hitting the Tab key will make the shell try to finish a partially typed word

Command structure

```
$ <command> <argument 1> <argument 2> <argument 3>
^     ^    ^     ^
|     |    |     |-- programs are provided these to
|     |    |         interpret (remember argc and argv[]?)
|     |    |
|     |    |-- words separated by whitespace
|     |
|     |-- certain things are actual programs, certain things
|         are handled by the shell ("built-ins")
|
|-- this is called a "prompt" and can take many forms
```

*nix and the filesystem

- As a spoiler for a future lecture, *nix exposes everything as a file
- Navigating through directories (folders) and interacting with files is a fundamental task
- We address and locate files via "paths"
- Each running program (including the shell) has a "current working directory"
- `/` enters/separates directories
- `.` refers to the current directory
- `..` refers to the "parent" directory (the directory that contains the current directory)

*nix and the filesystem

Types of paths:

- Absolute: starts with `/`
  - We call `/` the "root directory"; the starting point of the filesystem
  - `/home/brandon/Music/deemo-saika-rabpit.flac`
- Relative: starts from current or parent directory
  - `./dir1/dir2`
  - `../../some-dir`
  - Implicitly starts from the current directory if the path doesn't start with `/`, `.`, or `..`: `dir1/dir2`

Important commands

- `man`: "manual pages": gives info on programs
- `pwd`: "print working directory": tells you your current directory
- `ls`: "list": lists the contents of a directory
- `cd`: "change directory": changes your current directory
- `mv`: "move": moves files to another directory (actual moving) or another filename (renaming)
- `cp`: "copy": copies files
- `touch`: creates an empty file if one doesn't exist (otherwise updates its timestamp)
- `rm`: "remove": deletes files
- `grep`: searches files for data matches
- `cat`: "concatenate": technically concatenates files, often used to print out a file's contents
- Wikipedia has a nice list commands that *nix systems typically come with

Some common conventions

- Lots of commands/programs act on files
- A common pattern is `command path-to-file` e.g. text editors
  - `nano some-file.txt`
  - `vim some-code.cpp`
- `--help` as an argument is a common way to get info on how to use command
  - `cat --help`

Playing with output

- You can pipe output from command to another command with a pipe (`|`)
  - `echo "hello world" | rev`
- You can save output from a command to a file with a "redirection" (`>`)
  - `echo "hello world" > some-file`
- You can retrieve input from a file for a command with another "redirection" (`<`)
  - `rev < some-other-file`
- More to come in a future lecture ...

Intro to automation

- You can save a list of commands into a file
- This is known as a "script"
- You can now run this script whenever you want by invoking the filename as an argument for your shell of choice
  - `$ bash myscriptfile`
- This runs a new shell instance that runs each of those commands as if you had entered in the commands yourself
- If the file is marked as executable, you can also directly invoke it as a program
  - `$ ./myscriptfile`
  - Note you have to specify it as an explicit path (i.e. has a `/` present)
  - We'll discuss the specifics of this in a future lecture