Introduction and a dash of *nix

Lecture 1

Overview

1. Staff Introductions
2. Class Overview
3. Unix intro
4. Command line: what and why?

Staff Introductions

IA: Sowgandhi Bhattu

- CS-Eng + Entrepreneurship minor
- Interests include blockchain technology and data science
- Fun fact: Plays classical guitar

IA: John Paul O'Neill

- CS-LSA
- Interests include game development, free and open source software (FOSS), and increasing the accessibility of CS education
- Hobbies include biking, board games, and cooking

Instructor: Brandon Nguyen

- PhD Student under Prof. Scott Mahlke
- Undergrad at UT Austin in ECE
  - Primary focus: Computer architecture and embedded systems
  - Secondary focus: Digital signal processing
  - (spent my first two years as a BME doing premed...)
- Interests include computer architecture, compilers, and systems software
  - Weirdo who enjoys classes like 427, [2345]70, [34]73, 482, 583

Other stuff about me

- Modern and historical fencing (HEMA)
  - Shoutout to University of Michigan Fencing Club
- Slight addiction to Genshin Impact
- Trying to learn Vietnam's pre-romanization writing system
  - Turns out knowing written Chinese is a prerequisite

Course Overview

What is this class

- This class is for *anyone* wanting to become more effective at using their computer for development work
- This isn't necessarily a "tools" class
  - Tools come and go: does anyone remember COBOL and CVS?
- Each workplace will have its own tools and workflows
- The ultimate goal of this class is to help you learn to pick up, learn, and use new tools to solve problems
- The tools you learn along the way are the icing on the cake
- That being said, we will be focusing on Unix/Unix-like systems and shells in this class
  - Windows Command Prompt is not suitable for this class
  - Windows Subsystem for Linux (WSL) is suitable, however

Expectations

- Have a basic understanding of program control flow
  - e.g. if statements, loops, functions
- Have experience expressing your solutions in program statements
- Have some experience with a C or C++ or similar language
  - Let me know if you need help with the language itself
- Work is intended to be done alone
  - It can help to point each other to useful resources you find
  - Your code should be your own

Expectations

Technology

- Have a computer that runs Windows, mac OS, or Linux that you can install software on
- Chromebooks are welcome if they have Linux Beta (Crostini)
- Ubuntu 20.04 is going to be the reference environment for class
  - Most Linux distributions will have similar behvaior for things in this class, so don't fret if you're on 18.04 or Debian or whatever [i use arch btw]
- While most things can be done on mac OS, some tools have different behavior between Linux and mac OS
- Information will come out soon for getting access to an Ubuntu 20.04 environment
  - The release of M1 Macs has rocked the landscape and assumptions I can make...

Course communication

- Canvas: Announcements and a fancy gradebook
- Piazza: Course content and logistics related questions
- Discord: Casual and informal chat and questions
- Email: For personalized correspondance and more personal matters
  - Please start the subject line with "[EECS 201]" so I can find it

Course structure

Weekly lecture

- Attendance optional
- I will do live demos, mistakes can happen
  - Recovering from mistakes is always a learning opportunity
- Fill out a survey within a week of recording publication for extra credit
- There may be additional extra credit opportunities
- Feel free to "raise your hand" in Zoom or ask in chat
- I may record some supplementary lectures about certain smaller topics

Weekly "basic" assignment

- Guided light assignments to familiarize you with tools and what you can do with them
- Directly related to material covered in lecture

Course structure

Advanced component

- Less guidance than basic assignments
- May touch on some things not covered in lecture
- Provides practical experience in perusing documentation and applying what you know
- Can be fulfilled by doing 4 "advanced" assignments for full credit
  - Submitted online just like basic assignments
- Can also be fulfilled by doing a project
  - Checked out at an office hour
- Example projects:
  - Personal website
  - Picking up another language
  - Web scraping and data analysis
  - More to come...

Grading

- Two major grade categories: **Basic** and **Advanced**
- Basic has 60 total points
- Advanced has 40 total points
- Final score is the sum of these categories
  - There is no averaging: you just add numbers
  - Lecture extra credit is added on top
  - You can see how letter grades get assigned in the syllabus

Grading

Basic

- There will be 12 basic assignments worth 6 points each
- That means you only need to do 10 to get all 60 points
- The other 2 assignments serve as a buffer for you to miss/skip
- **Points past 60 are worth 50%: an 11th assignment would only be worth 3 points**
- If you do all 12 assignments:
  - 12 * 6 = 72 -> 60 + 12/2 = 66

Grading

Advanced

- There will be 12 advanced assignments worth 10 points each
- That means you only need to do 4 to get all 40 points
- You can also do *one* project for a total of 40 points
  - You can submit *one* partially completed project for partial credit
- **Similarly, points past 40 are worth 50%: an 11th assignment would only be worth 5 points**
- If you do all 12 assignments and the project...
  - 12 * 10 + 40 = 160 -> 40 + 120/2 = 100: no need to do basic assignments 😀

Grading

tl;dr you get points for each assignment and your letter grade is based on the total points

Any questions before we continue onto material?

Intro to *nix and the command line

First off, a poll

- Who has used a *nix environment?
- Who has Linux on their computer?
- Who has some sort of *nix on their computer?

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"
  - mac OS 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 mac OS
  - 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. mac OS 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, mac OS, 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

Demo

- This lecture was only a taste of the command line
- We will go more into depth on week 3
  - More about *nix
  - Control flow
  - Functions
- Next week will be Git 😀

Any further questions?

Addenda

Environment

- Terminal emulator: **Alacritty**
  - Former rxvt-unicode user until I learned how bad its font handling was
- Shell: **Zsh**
- Window manager: **i3-gaps**
- Compositor: **picom**
  - Does window transparency effects
- Notification server: **dunst**
  - Displays notifications

Software

- Editor: **Vim**
- System monitoring: **htop**
- Notetaking: **Xournal++**