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

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

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...)

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...)
- Weirdo who enjoys classes like 427, [2345]70, [34]73, 482, 583
    - Interests include computer architecture, compilers, and systems software

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...)
- Weirdo who enjoys classes like 427, [2345]70, [34]73, 482, 583
  - Interests include computer architecture, compilers, and systems software
- Non-tech hobbies include modern and historical fencing
  - Shoutout to University of Michigan Fencing Club

IA: Arav Agarwal

- CS-Eng and DS-Eng
- Interests include adversarial machine learning and reinforcement learning
- Hobbies include reading, coffee-making, and gaming (namely roguelikes)

IA: Sowgandhi Bhattu

- CS-Eng
- Interests include data analysis and web development
- Virtually interned at TD Ameritrade
- Fun fact: playing guitar since age 6

Course Overview

What is this class

- This class is for *anyone* wanting to become more effective at using their computer for development work

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 FORTRAN and CVS?

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 FORTRAN 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

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 FORTRAN 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

Expectations

- Have a basic understanding of program control flow
  - e.g. if statements, loops, functions
- Have experience expressing your solutions in program statements

Expectations

- Have a basic understanding of program control flow
  - e.g. if statements, loops, functions
- Have experience expressing your solutions in program statements
- Have a computer that runs Windows, mac OS, or Linux that you can install software on

Expectations

- Have a basic understanding of program control flow
    - e.g. if statements, loops, functions
- Have experience expressing your solutions in program statements
- 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)
    - Other Unix-likes/derivatives like FreeBSD are welcome too

Expectations

- Have a basic understanding of program control flow
  - e.g. if statements, loops, functions
- Have experience expressing your solutions in program statements
- 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)
  - Other Unix-likes/derivatives like FreeBSD are welcome too
- 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

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
- Feel free to "raise your hand" in Zoom or ask the Q&A

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
- Feel free to "raise your hand" in Zoom or ask the Q&A
- I may record some supplementary lectures about certain smaller topics
  - One is planned for virtual machines

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
- Feel free to "raise your hand" in Zoom or ask the Q&A
- I may record some supplementary lectures about certain smaller topics
    - One is planned for virtual machines

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

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

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

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

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:
    - Command-line based development environment (who needs a graphical IDE?)
    - Scripting Git
    - More details on this 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
    - Lecture extra credit is added on top
    - You can see how letter grades get assigned in the syllabus

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

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

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

Advanced

- There will be 12 advanced assignments worth 10 points each
- That means you only need to do 4 to get all 40 points

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

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 a project for a total of 40 points
  - You can submit a partially completed project for partial credit

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

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 a project for a total of 40 points
    - You can submit a 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...

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

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 a project for a total of 40 points
    - You can submit a 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 😀

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

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

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

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

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

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)

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?

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)

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)
- The *nix shells follows very similar basic syntax no matter what shell (bash, zsh, csh, etc.) you use

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)
- The *nix shells follows very similar basic syntax no matter what shell (bash, zsh, csh, etc.) you use
- *nix 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 *nix 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 command line 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)

Types of paths:

- Absolute: starts with `/` e.g. `/home/brandon/Music/saika-rabpit.flac`
  - We call `/` the "root directory"; the starting point of the filesystem
- 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`

Critical 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 or another filename
- `touch`: creates an empty file if one doesn't exist (otherwise updates its timestamp)

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`

Demo

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

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?