Lecture 01 Introduction

CS211 – Fundamentals of Computer Programming II

Branden Ghena – Winter 2022

Slides adapted from:
Jesse Tov

Northwestern

Welcome to CS211

- Course Goal: become a **better** and **broader** programmer

- First half
  - C programming
  - Unix shell

- Second half
  - C++ programming

- Introduces students to industry-standard languages and tools

- Builds foundational software design skills at a medium scale

Online classes

- We're online for the next two weeks, but I fully expect to be back in person after that
    - Note: I'm not an expert in public health

- For now, we'll make the best of being online

Branden Ghena (he/him)

- Assistant Faculty of Instruction
- Education
  - Undergrad: Michigan Tech
  - Master's: University of Michigan
  - PhD: University of California, Berkeley
- Research
  - Resource-constrained sensing systems
  - Low-energy wireless networks
  - Embedded operating systems
- Teaching
  - Computer Systems
    - CS211: Fundamentals of Programming II
    - CS213: Intro to Computer Systems
    - CS343: Operating Systems
    - CE346: Microprocessor System Design
    - CS397: Wireless Protocols for the IoT
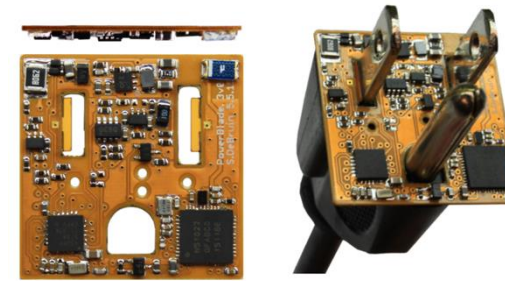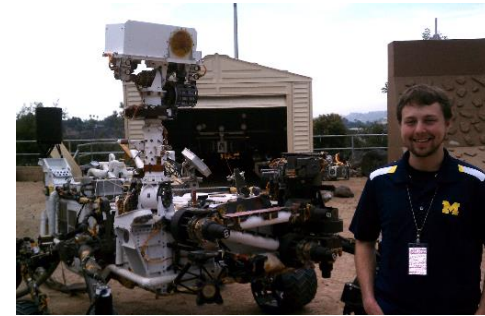
Things I love

Sruti Bhagavatula (she/her)

- Assistant Professor of Instruction
- Education
  - BS: University of Illinois Chicago
  - MS/PhD: Carnegie Mellon University
- Research
  - Spread of security and privacy information online
  - Measurement of security and privacy behavior
  - (Social networks, machine learning, IoT) + security/privacy
- Teaching
  - CS 211: Fundamentals of Programming II
  - CS 397/497: Data Privacy

My hobbies

Questions in class

- Please ask questions!!!
  - It's not just you who doesn't understand something.

- You can always ask questions verbally during class
  - Raise hand (in-person or virtually) or just speak up

- Bonus option: send messages in chat
  - Sruti will be watching and can answer if I don't
  - We'll have a chat system for questions when in-person too

Today's Goals

- Discuss **why** we teach (and require) this class

- Describe how this class is going to function

- Introduction to working in Unix shell (command line)

Outline

- **Why?**

- Course Overview

- Unix Shell

What does CS211 teach?

- **C and C++ Programming**

- Unix Shell

C - the most important programming language

- Old (1972), but nowhere near the first programming language
  - FORTRAN, LISP, ALGOL, COBOL, Basic, B, and many others came first

- Right time, right place, right capability
  - Enables both low-level control and (relatively) high level thinking
  - Fast, efficient, and highly portable

- Inspired everything that has come since
  - C syntax is copied partially or completely in MANY other languages
  - Lessons learned from using C inspired improvements to make programming easier

C++ - an evolutionary addition to C

- Additional features on top of C
  - Most important: classes to support Object Oriented Programming
  - Also includes a significant amount of libraries that C does not

- Enables more complicated software design
  - Manages which part of code can access which things at which times
  - Manages how things are named and referred to
  - Manages errors to help software respond to them

Things written in C/C++

- All major modern operating systems are partially or entirely C
  - Windows, Linux, MacOS, Android, iOS

- Scientific computing (mix of C and C++)
  - Mathematica, MATLAB, various scientific libraries

- Video game engines (often C++)
  - Unreal Engine, Unity, CryEngine

- Embedded control systems (usually C, occasionally C++)
  - Cars, Airplanes, Satellites and Rovers, Thermostats, Webcams, …

Upsides to C and C++

- You are in charge of everything
  - You can do anything you want without constraints

- Capable of directly interacting with hardware ("systems language")
  - Grab exactly as much memory as you need and manage it yourself
  - Makes it incredibly fast (~100x faster than Python)
  - Makes it incredibly efficient (no memory is wasted)

- These lead to the languages being very widely used
  - Top five programming languages for decades include C and C++

Downsides to C and C++

- You are in charge of everything
  - And nothing is taken care of for you

- Things you "can't" do are **UNDEFINED BEHAVIOR**
  - To enable portability, the languages just straight-up don't say what happens if you violate the rules
  - The computer could do *anything*

- Backwards compatibility means features are only ever added
  - You'll see this especially in C++, C just has less features total
  - C++ feels like a bunch of things stapled together
    - And there's an amazing programming language hiding in there

Analogies for programming languages

- Racket
  - Generic beginner's car that gets you places

- Python
  - Great car you can drive without a license
  - Unless you want to go really fast or on bad terrain, might be good enough

- C
  - A racing car that goes incredibly fast but breaks down every fifty miles

- C++
  - A souped-up version of the C racing car with dozens of extra features that only breaks down every 250 miles
  - But when it breaks down, nobody can figure out what went wrong

http://users.cms.caltech.edu/~mvanier/hacking/rants/cars.html

So why teach C and C++?

- You'll learn a lot more about programming
  - Syntax and ideas from C inspired a lot of other languages
  - Feels very different from Racket or Python

- You'll become a better programmer
  - You're going to run into a lot of errors and problems in this class
  - Hopefully they teach you to better design and plan your code

- Prepare you to dig deeper into computer systems
  - A "systems language" is needed to interact directly with hardware
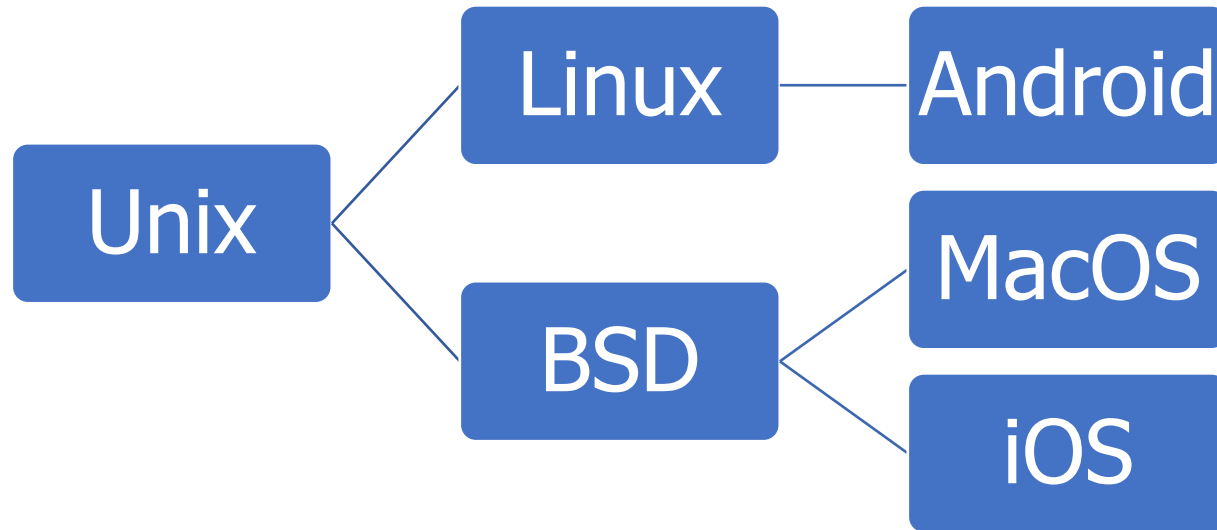 - Major options: Pascal, C, C++, Ada, Rust

What does CS211 teach?

- C and C++ Programming

- **Unix Shell**

Unix

- A wildly popular operating system in the 1970s and 80s

- Today refers to the *family* of operating systems inspired or grown from Unix
  - Particular design style for "everything is a file"
  - Various tools the OS is expected to provide
  - Command line interface, also known as a "shell"

C and Unix were born together

- Operating systems used to be written in assembly
  - Basic instructions specific to a certain processor family (see CS213)
  - So supporting a new computer type meant rewriting all of your software

- Unix development started in 1969 by Ken Thompson and Dennis Ritchie
  - Developed at Bell Labs, which was a computing research powerhouse

- C language was created in 1972 by Dennis Ritchie to write Unix programs
  - And they quickly rewrote the whole OS in C as well
  - This made the OS simpler to modify and easier to **port** to new systems
  - Unix became *enormously* popular due in part to its portability

Unix shell

- Text-based interface to a computer
  - Compare to graphical interfaces that need a mouse

- Necessary for remote interactions with many computers
  - Cloud servers
  - Specialized "headless" hardware

- Can be incredibly efficient and powerful
  - Find all JPEG files in this folder and change to be PNGs
  - `mogrify -format png *.jpg`

So why teach Unix shell?

- Many future classes are going to require you to work on a specialized computer that is shared by the class
    - More resources, specific capabilities, etc.

- Add another basic computing tool to your skillset
    - You might not use shell every day
    - But maybe you might

- You get to feel like a "hacker"
    - Shell isn't the only way to be a programmer, but is a stereotypical way

So, why CS211?

- It's going to make you a **much** better programmer

- It's going to teach you a bunch of new skills

- It's going to enable you to succeed in future classes

Architecture of a lecture

Outline

- Why?
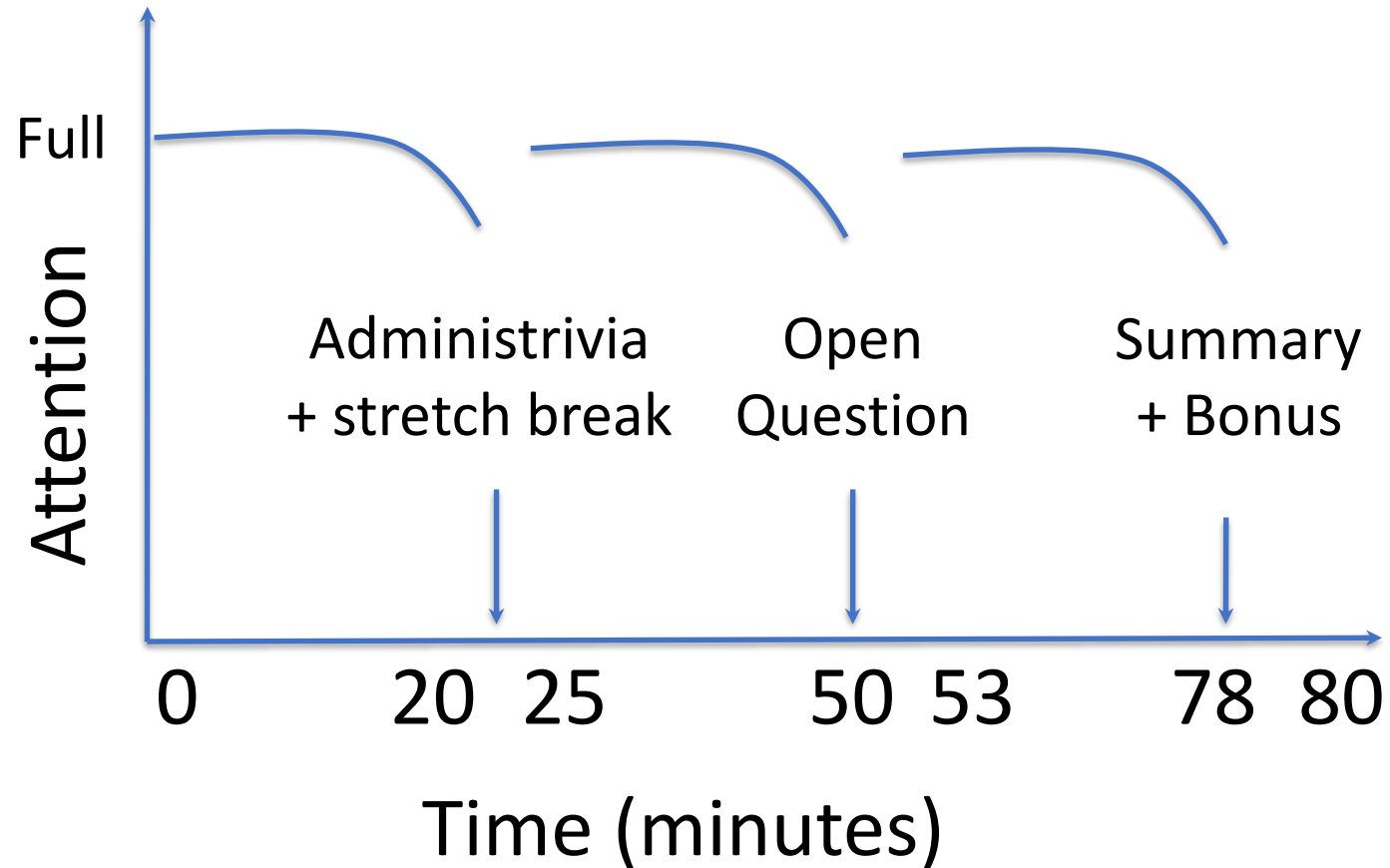
- **Course Overview**

- Unix Shell

Course Staff

- TA (1)
  - Sherwin Shen
    - PhD student in Computer Science

- PMs (12)
  - AJ Hesby   Alexander Redding
  - Antonio Rocha   Brian Gleason
  - Chris Song   Danche Smilkova
  - Dilan Nair   Eli Barlow
  - John Sanchez   Mirage Modi
  - Naythen Farr   Nick Baird

- Their role: support student questions via office hours and campuswire

Lectures

- Lectures: synchronous, recorded via Zoom (for now)
  - Please attend and ask questions!
  - Panopto tab on Canvas will have recordings (a few hours later)
    - This should still be true when we're back in-person

Labs

- Small, guided practice sessions to help you learn
  - Teach you a new skill/language

- Two parts
  - **Lab guide** will walk you through doing some things
  - **Lab assessment** on Canvas will ask a few short questions about it
    - Should be easy if you did the lab

- These are not formal assignments or quizzes
  - You may work with others on them

Quizzes

- Multiple quizzes instead of a big exam
  - Should be four total
  - Each is roughly 15-20 minutes

- Quizzes cover mainly material from the last two weeks
  - But build upon knowledge from the entire course

- More details on these to come
  - We will hopefully be back in person for the first one…

Homeworks

- Programming assignments with about a week to complete it
  - This is where you'll learn the most in class

- First four are C, last two are C++

- Homeworks 1 and 5 are on your own

- Other homeworks are with a partner of your choosing
  - We'll put up a survey for those who want to be paired with a random partner

- These are serious. Be careful about academic integrity on these

Final Project

- A bigger homework, where you get to choose what you want to do
  - Done with a partner of your choosing

- Make an "interactive program" (usually a game)
  - Examples: Pacman, Tetris, Two-dots, Checkers, Desert Bus

- This is your chance to do something interesting and fun!

- Can be a significant amount of work though

Grade composition

Category	Count	Total Value
Labs	5	5%
Quizzes	4	10%
Homework	6	60%
Final project	1	25%

Relative homework difficulties

Homework	Difficulty
Hw01	2
Hw02	5
Hw03	7
Hw04	11
Hw05	6
Hw06	9
Final Project	10ish*

Hw04 is the last in C
Then a one week break
Hw05 is the first in C++

* But really it's up to you

Late Policy

- You can submit *homeworks* late
  - Quizzes and labs cannot be submitted late

- 10% penalty to maximum grade per day late
  - Example: three days late means maximum grade is 70%

- Final project has a sliding scale
  - 90% for up to 24-hours late
  - 60% and 30% for the two days after that

Slip Days

- Slip days let you turn in a homework late and receive no penalty

- Each student gets **3 slip days**
  - Apply to **homeworks only** (not final project or labs)
  - You don't need to tell us you're using them, we'll just automatically apply them at the end of the year
  - Be sure to coordinate about them on partner assignments

- Examples:
  - Turn in hw01 three days late
  - Turn in hw04 two days late and hw06 one day late
  - Turn in hw02 four days late with only a one-day penalty

We can support you for unexpected problems

- We can be flexible with deadlines for problems outside of your control
  - Sick, family emergency, broken computer
  - Contact us (via Campuswire)

Collaboration in CS211, three levels:

1. **Partner Collaboration**
    - Your code and the other student's code are identical because you share it and work on it together
    - ONLY for registered partners on specified homeworks

2. **Close Collaboration**
    - You communicate about code however you see fit
    - ONLY acceptable for labs

3. **Arms-Length Collaboration**
    - You discuss problems and solutions at a high level
    - MAY NOT read, write, look at, record, or transcribe code
    - MAY NOT have the code up on screen during collaboration
    - MUST cite your sources, both arms-length collaborators and other resources

Refer to syllabus for the official version of this policy

Academic Honesty

- In CS211, we take cheating very seriously

- Cheating is when you:
  - Engage in an inappropriate level of collaboration
    - Such as look at another student's code

  - Enable another student, *present or future*, to cheat
    - Such as letting a CS211 student read your code next year

  - Fail to cite your sources (friend, Stack Overflow, etc.)
    - Such as you get a big hint and don't acknowledge where it came from in a code comment

Academic Honesty

- **Please do not cheat in CS211**

1. If you don't write code, you won't learn!

2. Cheating on code is super easy to catch!!
   - No, like really really easy
   - All suspected cheating is reported to the relevant dean for investigation
   - Last time I taught it was 8 cases

- If you are unsure about a situation, ask the staff on Campuswire

Getting Help – Campuswire

- Post questions here
  - Staff and I monitor and answer questions
  - You can also answer each other! (or note that you have the same issue)

- I'll also post useful or interesting notes here

- Do NOT email me. Post to Campuswire instead
  - I won't see your email until way later and then I'll feel guilty about it

Getting Help – Office Hours

- Office hours are mostly hosted by the PMs and TA
  - Sruti and I will have some too! Especially for higher-level questions

- Schedule
  - We're going to host a TON of office hours
  - Entirely remote for now (on gather.town)
    - Hopefully we can add some in-person hours later
  - Details to follow, schedule on Canvas homepage

- Reminder: office hours are meant to augment the class
  - Attend them!

Getting Help – Request a Meeting

- Lecture is my side gig

- My main job is helping students succeed

- If you are struggling, reach out and I will meet with you
    - Course material
    - Homework
    - Other stuff going on in your life

Advice

- Submit assignments early and often!

- If you find this course difficult, that's because it **is** difficult.

- However, nobody fails unless they give up.

- You belong here and can succeed here.

- **Be kind to each other.**

Break + relevant xkcd

Outline

- Why?

- Course Overview

- **Unix Shell**

How do you get a Unix shell?

- Have a MacOS or Linux computer
  - Or set up Windows Subsystem for Linux (WSL) on Windows

- Install Virtualbox and Linux
  - Installing Ubuntu is free and only takes twenty minutes

- Log in to a class server remotely!
  - This is what we'll do for CS211
  - Lab01 teaches you how to do this (posted later today)

Command line interfaces

- Text-based commands

- Positives
  - It's easy to be precisely clear about what you want and how things are configured

- Negatives
  - How do you remember everything?

- Reality
  - There will be a few dozen commands you'll memorize (after practice)
 - And you'll learn how to look up everything else

Live code demo!!!

Commands for moving between directories

- Directory structure and moving through it
  - `ls`
    - Lists files in the current directory
  - `cd`
    - Change directory
  - `pwd`
    - Prints the path of the current directory

- Mis-typing something
  - "Command not found" means you tried to run something invalid
  - `fish:` <u>`somecommandyoumistyped`</u>`: command not found...`

Directory structure in Linux
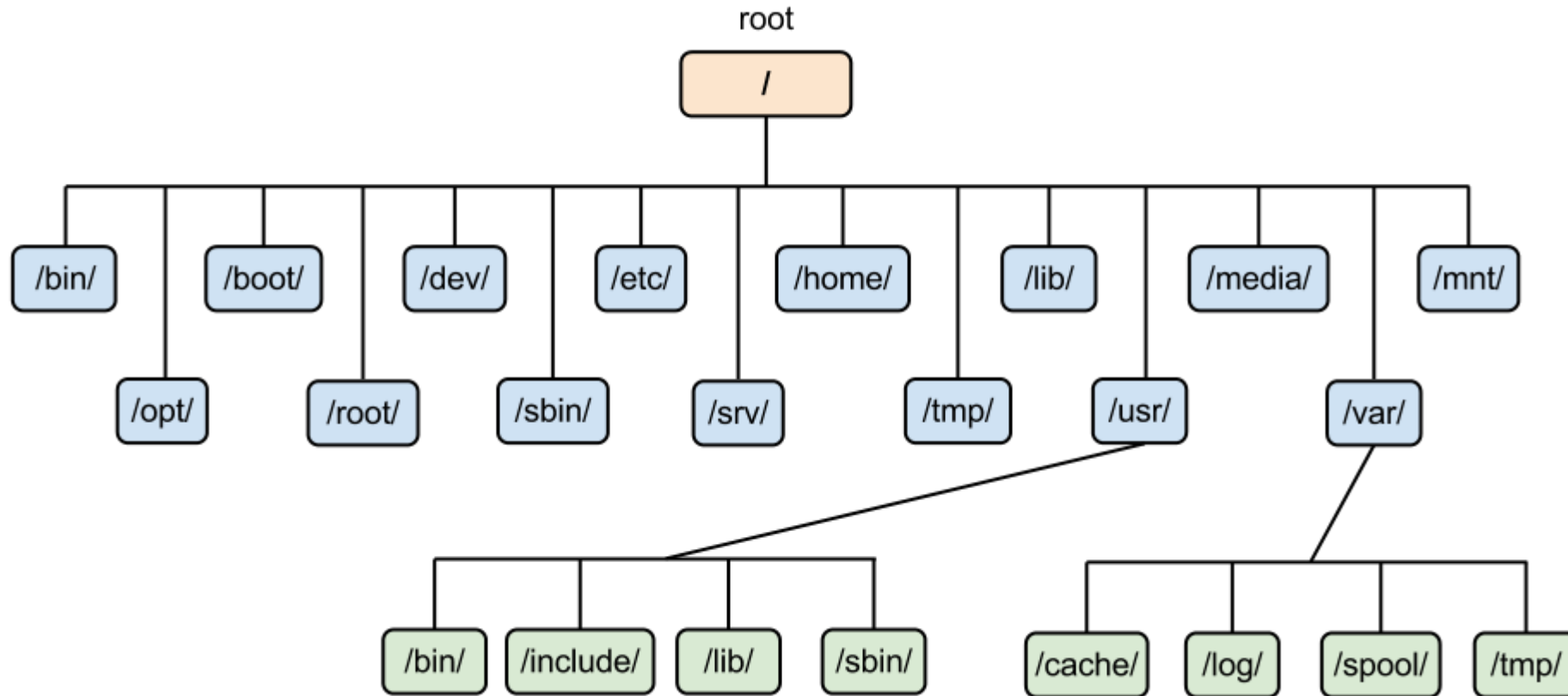
- Example: `/usr/bin/` is the path to user-installed programs

Special paths

`.`　　　　　　　the current directory
`..`　　　　　　the parent of the current directory
`../../`　　　the parent of the parent of the current directory
`../../../`　and so on...

`-`　　　　　　 the previous directory you were in before the current one

`~/`　　　　　 the home directory of the current user (your home)
`~cs211`　　 the home directory of the user cs211
　　　　　　　　(works for any user, but you'll probably won't interact with other users)

`/`　　　　　　 the root directory (analogous to `C:\` on windows)

Relative vs absolute paths

- Relative paths are relative to the current directory
  - `../`
  - `src/`
  - `../../code/src/../build/`

- Absolute paths have the full path name to the location
  - `/home/branden/`
  - `/home/branden/cs213/code/`
  - `/home/branden/cs213/code/src/../build/`

Wildcard in path names

- Sometimes you're not sure exactly what the name is
  - Or there might be multiple files that you want to interact with simultaneously

- The wildcard symbol, *, replaces any number of characters in a path name

- Examples
  - `ls /home/*/`   List all files in all user's home directories
  - `ls ~/cs21*/`   List all files in any directory starting with cs21
  - `ls code/src/*.c` List all files that end with ".c" in code/src/

Tab Completion

- Typing takes toooooooo loooooooonnnnggg
  - Solution, let the computer guess what you're trying to type

- Pressing tab while part-way through typing just about anything in terminal will tab-complete it for you
  - As long as you have typed enough characters so that only one option remains, it will complete it
  - If multiple options remain, it will stop trying

Command flags

- `man`
  - Opens the manual pages for a program
  - Example: `man ls`

- Flags are configurations for a command that change what it does
  - `ls -l` lists files in the current directory in a vertical list with details
  - `ls -t` sorts the ls output by most recently modified
  - `ls -l -t` does both

- You can type multiple flags after a single dash
  - `ls -lt` is equivalent to `ls -l -t` is equivalent to `ls -tl`

Searching for things

- `grep –r "text" *`
  - Explanation
    - Grep prints lines matching a pattern
    - The pattern in this case is "text"
    - `–r` means search recursively, i.e. in this directory and all subdirectories
    - * means to search in any file in the current directory
  - Summary
    - Search all the files here and below for the word "text"

Working with files

- `cat path`
  - Prints out the contents of the file

- `mv path1 path2`
  - Moves a file from path1 to path2

- `cp path1 path2`
  - Copies a file from path1 to path2

- `rm path`
  - Deletes (removes) a file

Editing files

- There are many different terminal text editors
  - And there are holy wars about why one is *best*
  - **There is no best. Just use whatever you like**

- Example editors
  - Vim, Emacs, Nano

- In CS211, I'll be teaching you using the Micro text editor
  - Occasionally I'll open vim by accident. Someone yell at me when I do
  - https://micro-editor.github.io/

Editing with Micro

- micro filename
  - Opens micro, editing filename

- Works just like any text editor you've used
  - Mouse moves the cursor around, as do the arrow keys
  - Typing makes text appear
    - (This isn't true in some shell editors, _{looking at you vim})

  - Ctrl-s   save the file
  - Ctrl-o   open a file
  - Ctrl-q   quit

Build a C file if there's enough time

- Lab01

- https://nu-cs211.github.io/cs211-files/lab/lab01.pdf

Don't be overwhelmed!!!!

- You have plenty of time to learn this

- Lab01 guides you through the same kinds of commands I did today, step by step

- Practice is the only thing that will really help
  - And CS211 will give you plenty of practice

Helpful guides

- Great lecture notes on using the shell
  - https://swcarpentry.github.io/shell-novice/

- Tool to explain various shell commands
  - https://explainshell.com/

- Tool to explain how to use various shell commands
  - Just type the command into the box at the top
  - https://tldr.ostera.io/

Outline

- Why?

- Course Overview

- Unix Shell