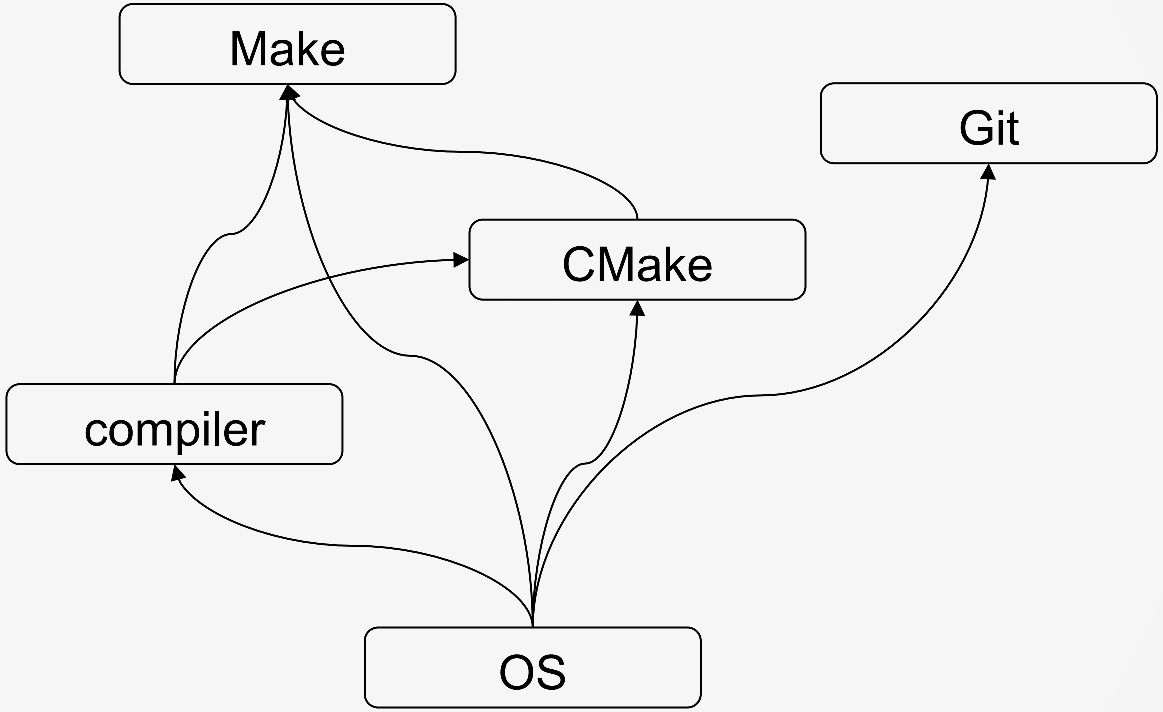
WASHINGTON STATE UNIVERSITY

(1 - 1) Intro to basic tools

Motivation

- Basic tools:
  - Linux (operating systems, OS)
  - G++/gcc (compilers)
  - Make (build automation tool)
  - Cmake (cross-platform build file generator)
  - Git (version control system)
- Goal: basics for completing assignments in this course

Motivation

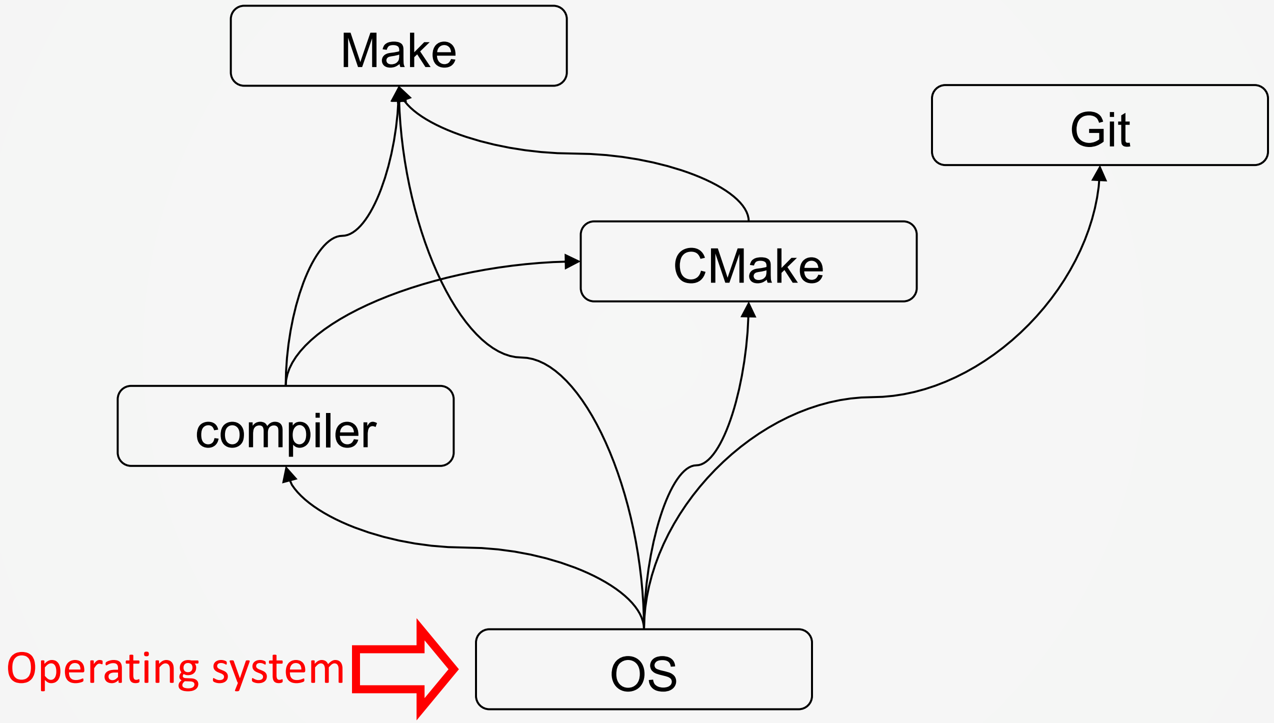


OS + compiler → **executable** program

OS + compiler + Make → **automation** of executable program

OS + compiler + Make + Cmake → **Cross-platform** automation of executable program

Motivation



OS + compiler → **executable** program

OS + compiler + Make → **automation** of executable program

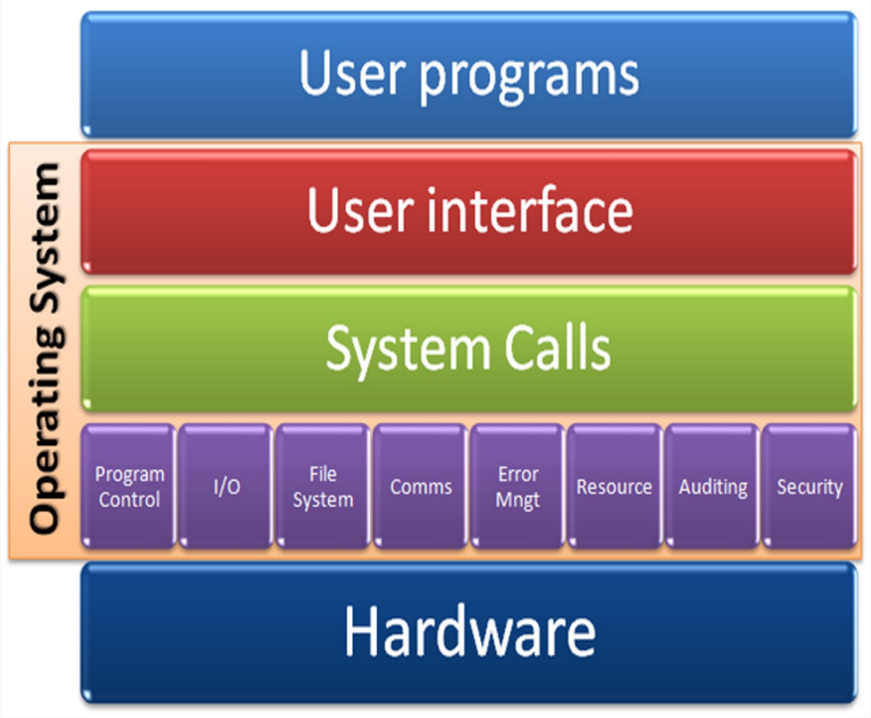
OS + compiler + Make + Cmake → **Cross-platform** automation of executable program

What is OS?

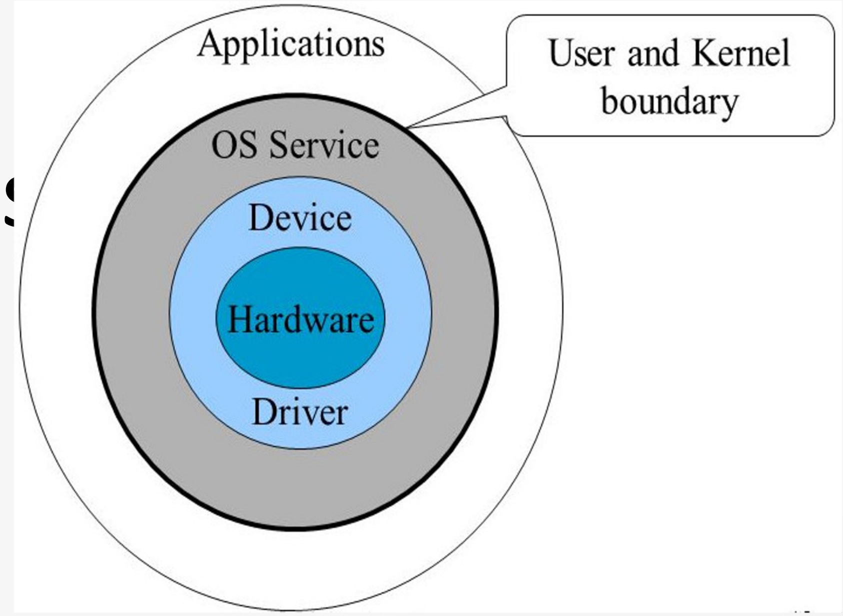
Short answer: It’s a program

OS

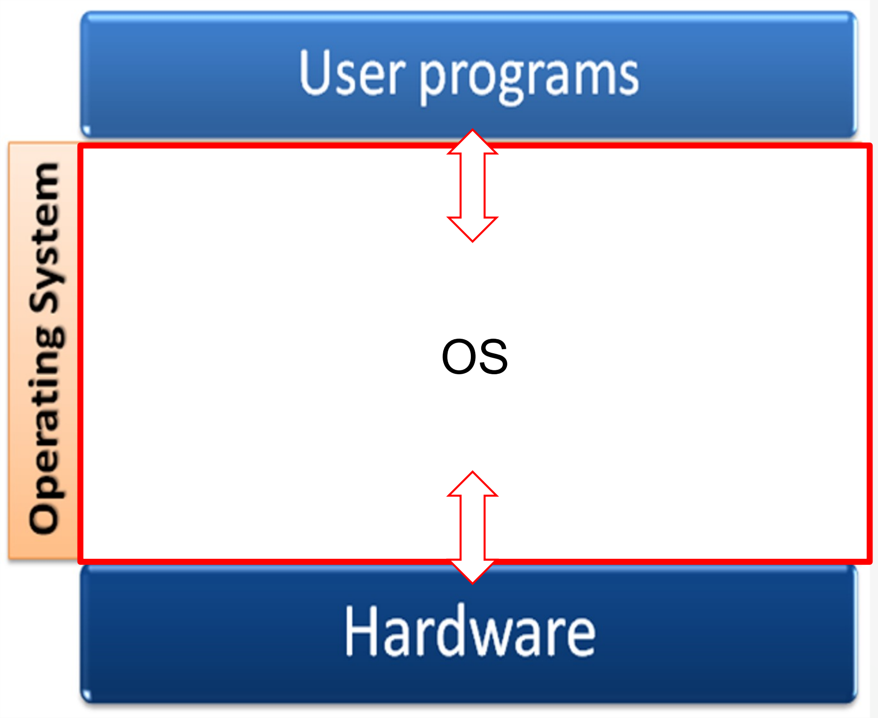
What is OS?

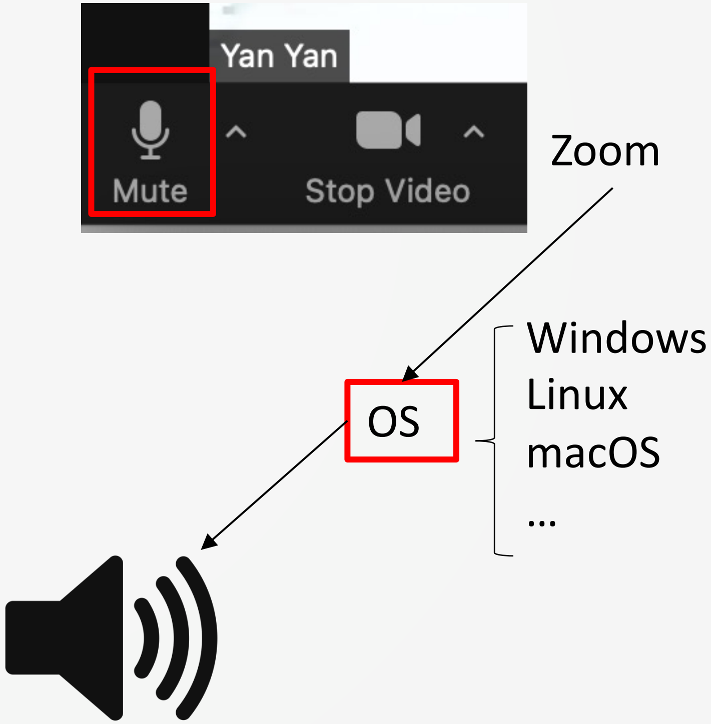


wer: It’



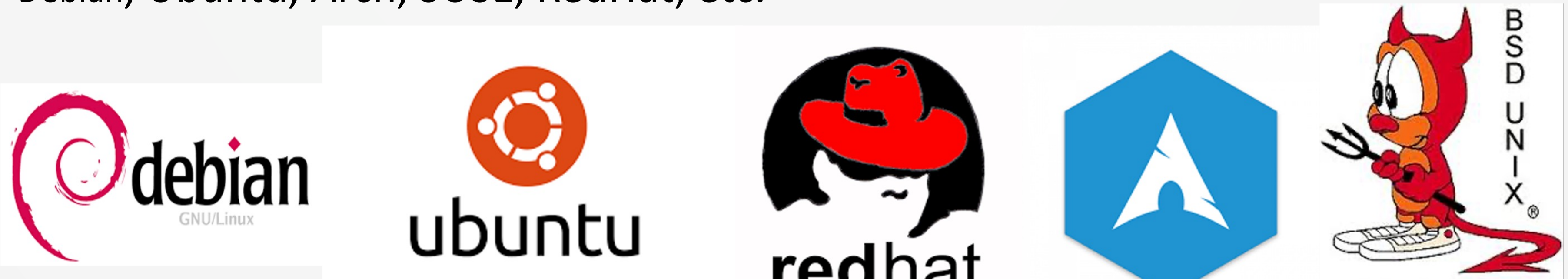
What is OS?


User programs
Operating System
OS
Hardware


Yan Yan
Mute
Stop Video
Zoom
OS
Windows
Linux
macOS
…

Officially “Linux” is just a kernel

- Linux is only the operating system kernel
- Colloquially, it’s used to refer to a whole distribution with user tools
- The result is the GNU/Linux name, since the user tools are (mostly) GNU
- Packaged together they become Linux distributions or “distros”
  - Debian, Ubuntu, Arch, SUSE, RedHat, etc.



The original BSD daemon appeared first in 1983 on the cover of the 4.2BSD manuals published by the Usenix Association

LOTS of OS

- Linux: Unix-like
- BSD (freeBSD, OpenBSD, NetBSD): Unix-like
- macOS (Darwin, from BSD): Unix-like
- WinNT
- Plan9

https://en.wikipedia.org/wiki/List_of_operating_systems

UNIX philosophy?

- https://en.wikipedia.org/wiki/Unix_philosophy
The Unix philosophy, originated by Ken Thompson, is a set of cultural norms and philosophical approaches to **minimalist, modular software development**
- Emphasis on building **simple, short, clear, modular, and extensible code** that can be easily maintained and repurposed by developers other than its' creators

GUI v.s. Command Line

VT100 terminal

- The eternal question (since about 1973)
- Both are great!
- But… since all data are strings in the end, the **command line** has more power for the user to interact with the OS
- Read “In the Beginning Was the Command Line” by Neal Stephenson
    - http://faculty.georgetown.edu/irvinem/theory/Stephenson-CommandLine-1999.pdf
    - Highly recommended for understanding UNIX and Linux philosophy

Why Linux?

- Probably 2 primary reasons: Control & Utility
  - You have **control** over your system (at ALL levels)
  - It has a large suite of extant **tools** for most uses
- Linux can be used for almost any computing environment
  - The Kernel scales very well, and you can use/edit the source to suit your needs
- Linux/UNIX was designed for **remote** access by default
  - GUIs aren’t great over the network.
  - UNIX was built for **multi-user and multi-tasks** and Linux inherited that powerful structure

To Summarize OS

- An OS **manages** and mediates the hardware in your computer
- An OS **launches** other programs and schedules them
- An OS **manages** memory and disk use
- The user starts in a shell (GUI or command line), which launches other applications as needed

Quick VirtualBox intro

- VirtualBox is a program to run operating systems on other operating systems!
- These are called virtual machines
  - actually, the hardware is virtual, not the OS
- Can use it for testing OSes, virtual networks, trying different tools out, etc.

Options for Linux access

- Install Windows Subsystem for Linux (**WSL**)
- MacOS is also OK
- Install Linux on a computer, either solely or dual boot
- Install VirtualBox (or something else) and make a Virtual Linux machine (*can be slow*)

Common programs

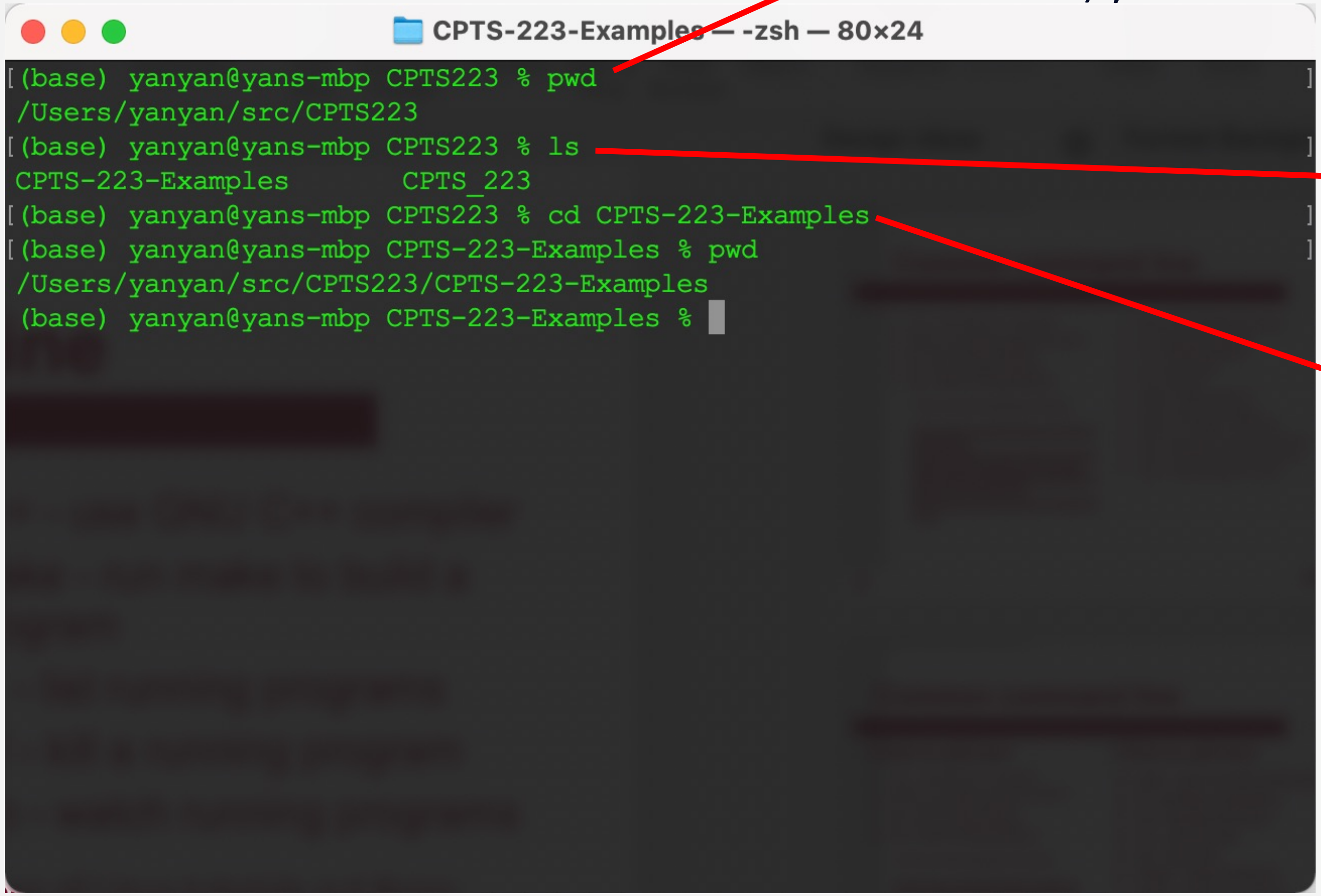
- pwd – print working directory
- ls - list files in directory
- cd - change directory
- rm - remove file
- cp - copy file
- mkdir - make directory
- rmdir - remove directory
- nano / vi / emacs - edit a file
- ssh - use ssh to connect to server
- scp - copy file over ssh to server
- man - manual page for tools
- g++ - use GNU C++ compiler
- make - run make to build a program
- ps - list running programs
- kill - kill a running program
- top - watch running programs

Tons of Linux tutorials out there:

https://www.geeksforgeeks.org/linux-commands/
https://ryanstutorials.net/linuxtutorial/
http://linuxcommand.org/index.php
https://www.codecademy.com/learn/learn-the-command-line
http://www.ee.surrey.ac.uk/Teaching/Unix/

OS

Common programs



```
(base) yanyan@yans-mbp CPTS223 % pwd
/Users/yanyan/src/CPTS223
(base) yanyan@yans-mbp CPTS223 % ls
CPTS-223-Examples        CPTS_223
(base) yanyan@yans-mbp CPTS223 % cd CPTS-223-Examples
(base) yanyan@yans-mbp CPTS-223-Examples % pwd
/Users/yanyan/src/CPTS223/CPTS-223-Examples
(base) yanyan@yans-mbp CPTS-223-Examples % 
```

pwd: displays the directory (or folder) you're currently in

ls: lists down all the directories and files inside the present working directory or specified directory

cd: move a directory/folder

Grab a Cheat Sheet

- The world of UNIX commands is large. As you’re starting out, grab a cheat sheet and even **keep a notepad of commands you’ve used** until you’re more comfortable with the tool set.
- Here’s a pretty reasonable one:
  - https://files.fosswire.com/2007/08/fwunixref.pdf
- Another source for commands:
  - https://www.geeksforgeeks.org/linux-commands/

How to run commands

- On the command line, the first thing you type is the **name of a program** to run. If it's not a **standard program**, you also need to have **the path to the file**
- Everything after the name of the program are command line **options**
  - Unless you chain multiple programs together with pipes or give shell I/O redirections
- Command line options tell the program what you want it to do
  - **ls** (lists the files and directories) … **ls -la** (lists all including hidden, plus other stuff)
  - The man page (**man ls**) will tell you more of what's available for a tool
- Eventually, as you gain experience, these commands start happening really quickly and usually without much conscious effort.

How to run commands (more)

- Your home directory is also called: ~
  - $HOME is the variable (the shell is a coding environment, right?) holding it too
- Filesystem norms: /home, /etc, /usr, /dev, /var, /tmp, /mnt, /opt, /root
- Can this class be done on a Raspberry Pi computer? Yes!
  - Raspbian is a debian fork, just FYI
- There are various **shells**, but most people use bash

Command line options

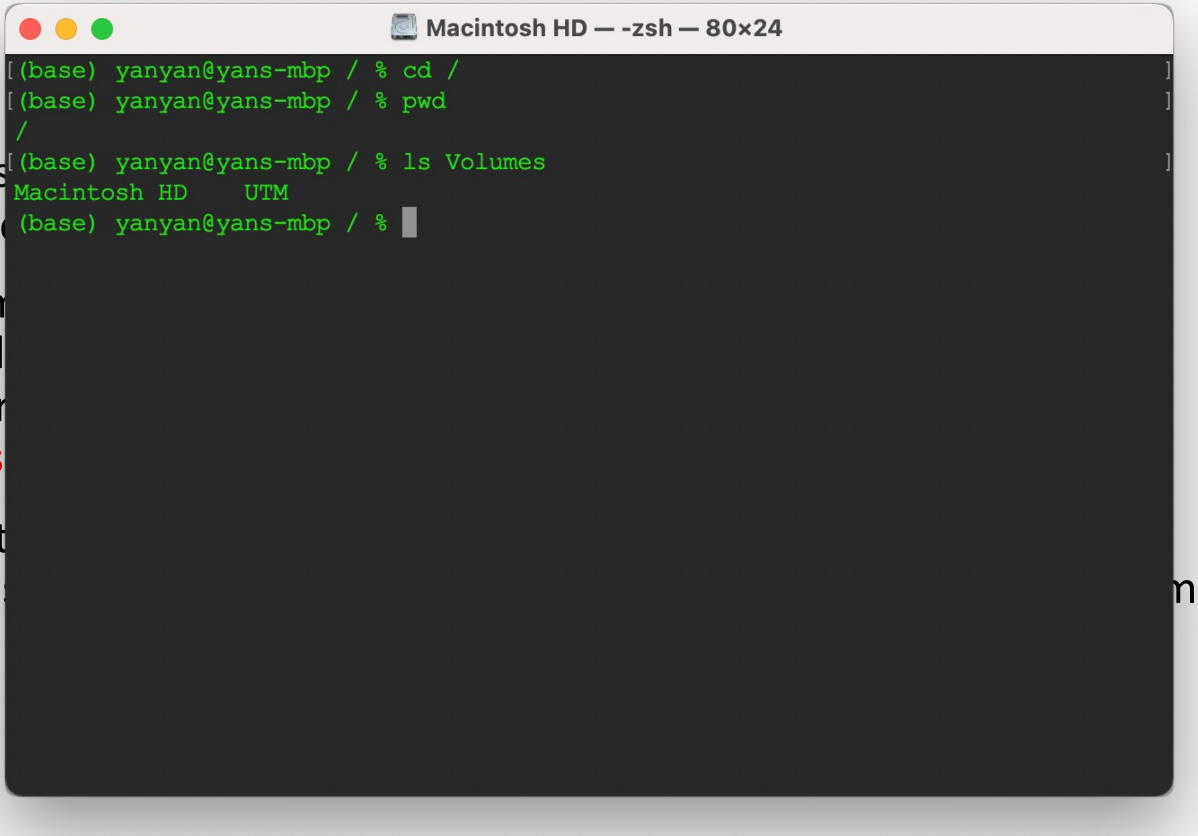
- Remember how your programs would sometimes start with:
  int main(int argc, char* argv[])
- Yeah, argc and argv are set by the command line options
- **argc** is the number of strings (split by spaces) the program was run with
- **argv** is an array of char* strings, one with each “word”
- argc is always **at least 1** since the first string is the name of the file used to run the program, including the path
- GUI IDEs (VS) have ways to set the options passed while testing builds

UNIX filesystem structure

- Where's C:\? – Windows! Not part of this system!
- Everything lives in a single tree **under /**
  - This is called "slash" or root (not to be confused with the root user)
- More filesystems (disks, etc) are just mounted **under / somewhere**
  - Command to add a disk is: mount Removing is: unmount
  - All disks are in the devices directory: **/dev**
  - Ex: **/dev/sda1**
- Most of this is taken care of for you in a default Linux install from a distro
  - But if you start using thumb drives or adding hard drives to your system, this shows up

UNIX filesystem structure

- Where's C:\? –
- Everything lives
  - This is call
- More filesystem
  - Command
  - All disks ar
  - Ex: **/dev/s**
- Most of this is t
  - But if you
  m, this shows up



```
[(base) yanyan@yans-mbp / % cd /
[(base) yanyan@yans-mbp / % pwd
/
[(base) yanyan@yans-mbp / % ls Volumes
Macintosh HD    UTM
(base) yanyan@yans-mbp / % 
```

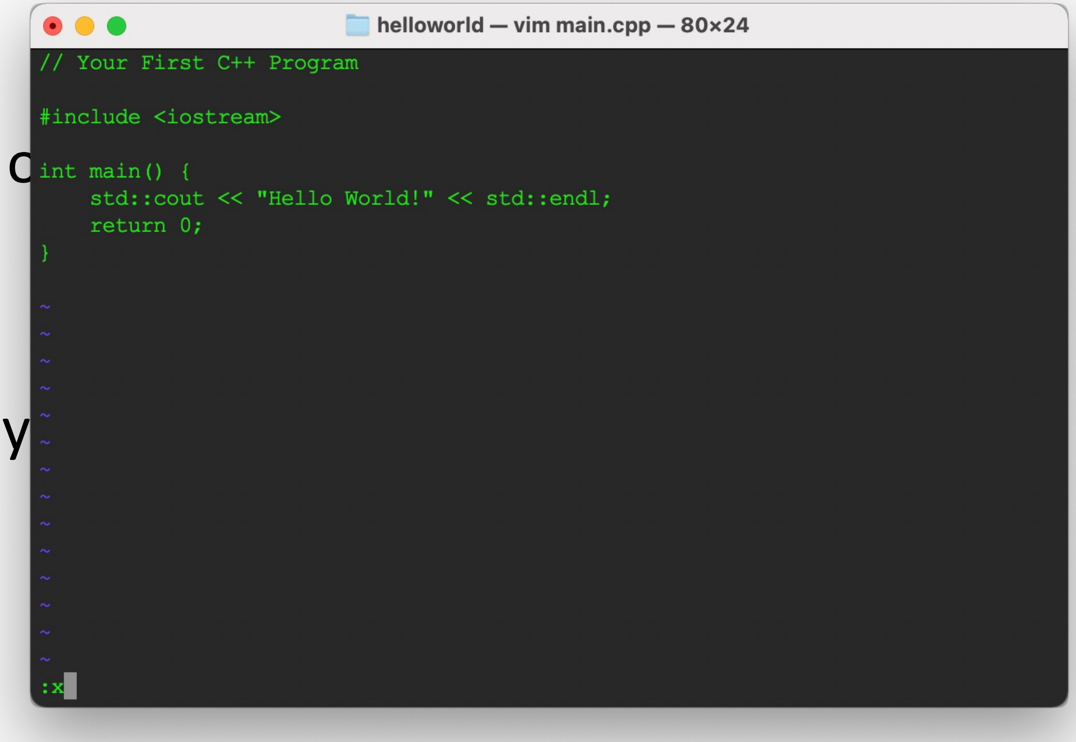
Editing files

- The big three options:
  - Vi
  - Emacs
  - Nano
- There is plenty more options, but these are the big 3

OS

Editing files

- The big three c
  - Vi
  - Emacs
  - Nano
- There is plenty



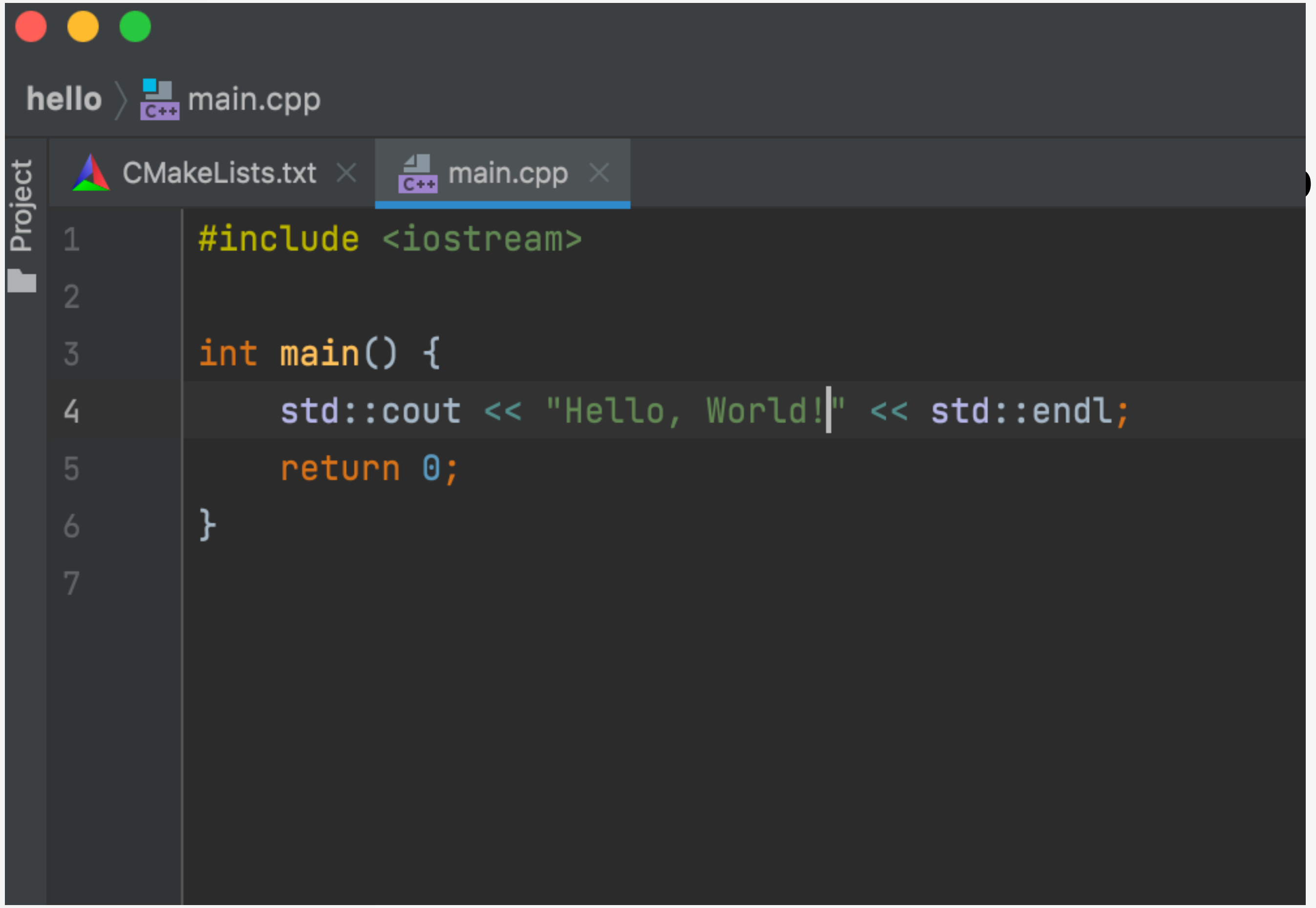
```
// Your First C++ Program

#include <iostream>

int main() {
    std::cout << "Hello World!" << std::endl;
    return 0;
}

~
~
~
~
~
~
~
~
~
~
~
~
~
~
:x
```

GUI IDE options

- If you’ve got a desktop, there’s options for GUI tools
  - CLion
  - netbeans
  - Code::Blocks
  - KDevelop
  - Eclipse
  - CodeLite IDE
  - Geany IDE
  - Vscode
  - …

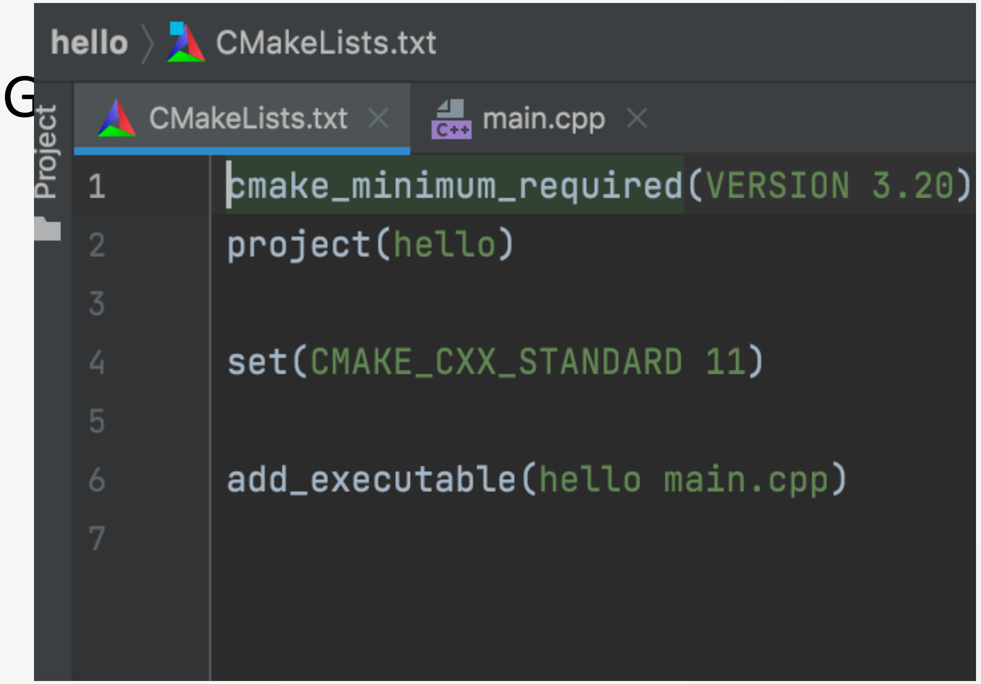
OS

GUI IDE options



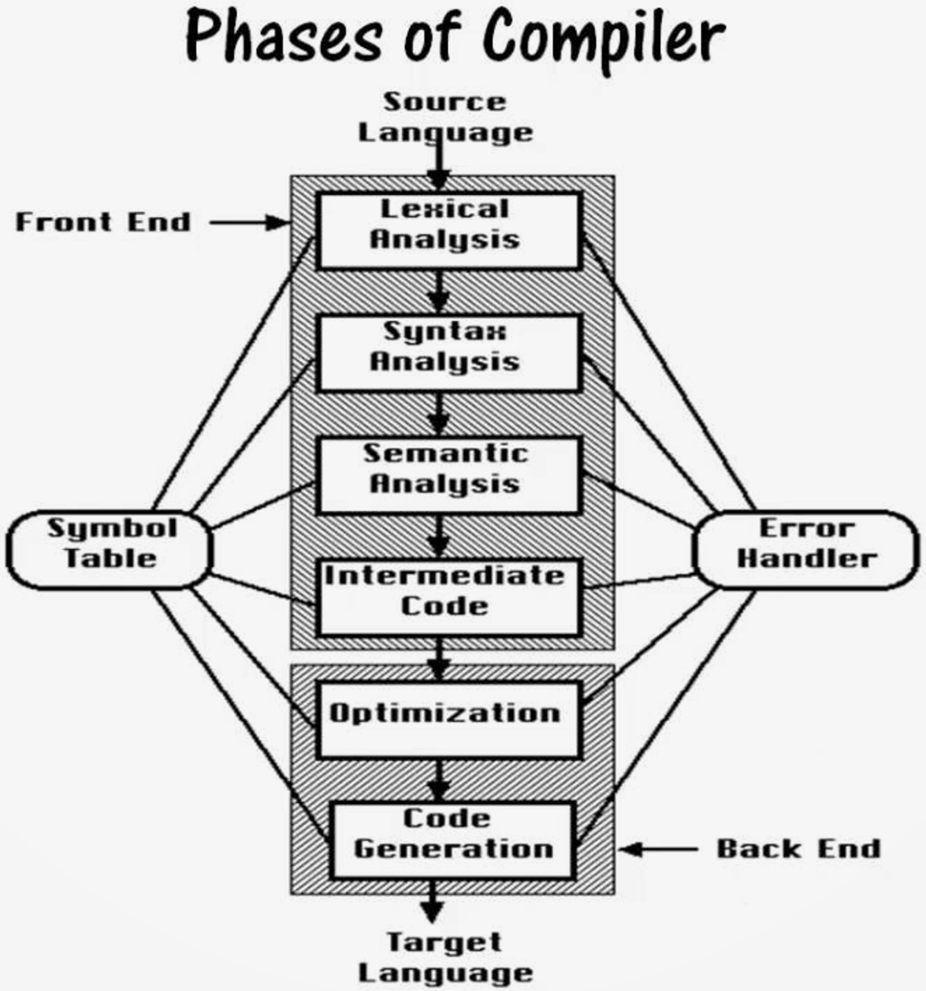
```cpp
#include <iostream>

int main() {
    std::cout << "Hello, World!" << std::endl;
    return 0;
}
```

r G



```cmake
cmake_minimum_required(VERSION 3.20)
project(hello)

set(CMAKE_CXX_STANDARD 11)

add_executable(hello main.cpp)
```

What is a compiler?



What is a compiler?

- What is a compiler?
  - It is a program! (or several programs that work together)
  - It is a very special program - it has to take itself as input!
  - It takes a string of text and converts it to a different string of text
  - It takes a string in one language and converts it to another language:
    - C → Assembly → Machine code
    - Java → Byte Code
- Visual Studio has a compiler within the IDE called Visual C++
  - You’ve used it every time you “built” your programs

What is a compiler?

- What is a compiler?
  - It is a program! (or several programs that work together)
  - It is a very special program - it has to take itself as input!
  - **It takes a string of text and converts it to a different string of text**
  - **It takes a string in one language and converts it to another language**:
    - **C → Assembly → Machine code**
    - **Java → Byte Code**
- Visual Studio has a compiler within the IDE called Visual C++
  - You’ve used it every time you “built” your programs

LOTS of compilers

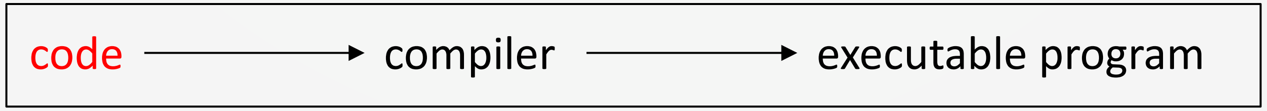
- Visual C++
- **GNU gcc/g++**
- Clang C/C++ (LLVM)
- Intel C Compiler
- Python interpreter
- A huge list: https://en.wikipedia.org/wiki/List_of_compilers
- To make a programming language useful (beyond a spec), you'll need to build a compiler for it.

This semester

Programming process

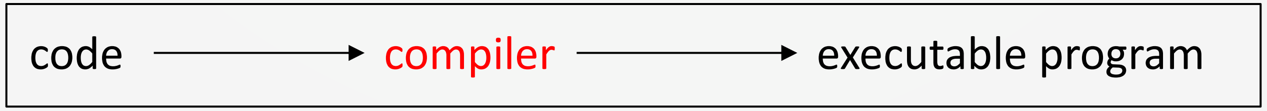
- Create a string in a given language
- Pass that string to a compiler
- Take results from compiler and execute those

- You’ve been doing this all along inside of VS IDE, but here it is going to be more explicit:
  1) Edit a text file (or more text files)
  2) Pass that text file to g++
  3) Run resulting file as an executable program



Programming process

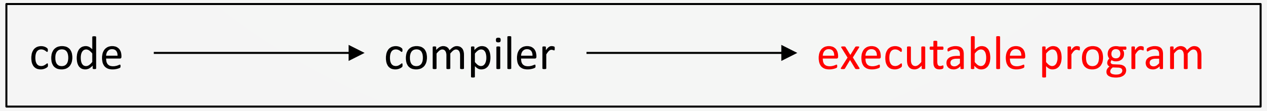
- Create a string in a given language
- Pass that string to a compiler
- Take results from compiler and execute those
- You’ve been doing this all along inside of VS IDE, but here it is going to be more explicit:
  1) Edit a text file (or more text files)
  2) Pass that text file to g++
  3) Run resulting file as an executable program



code → compiler → executable program

Programming process

- Create a string in a given language
- Pass that string to a compiler
- Take results from compiler and execute those
- You’ve been doing this all along inside of VS IDE, but here it is going to be more explicit:
  1) Edit a text file (or more text files)
  2) Pass that text file to g++
  3) Run resulting file as an executable program



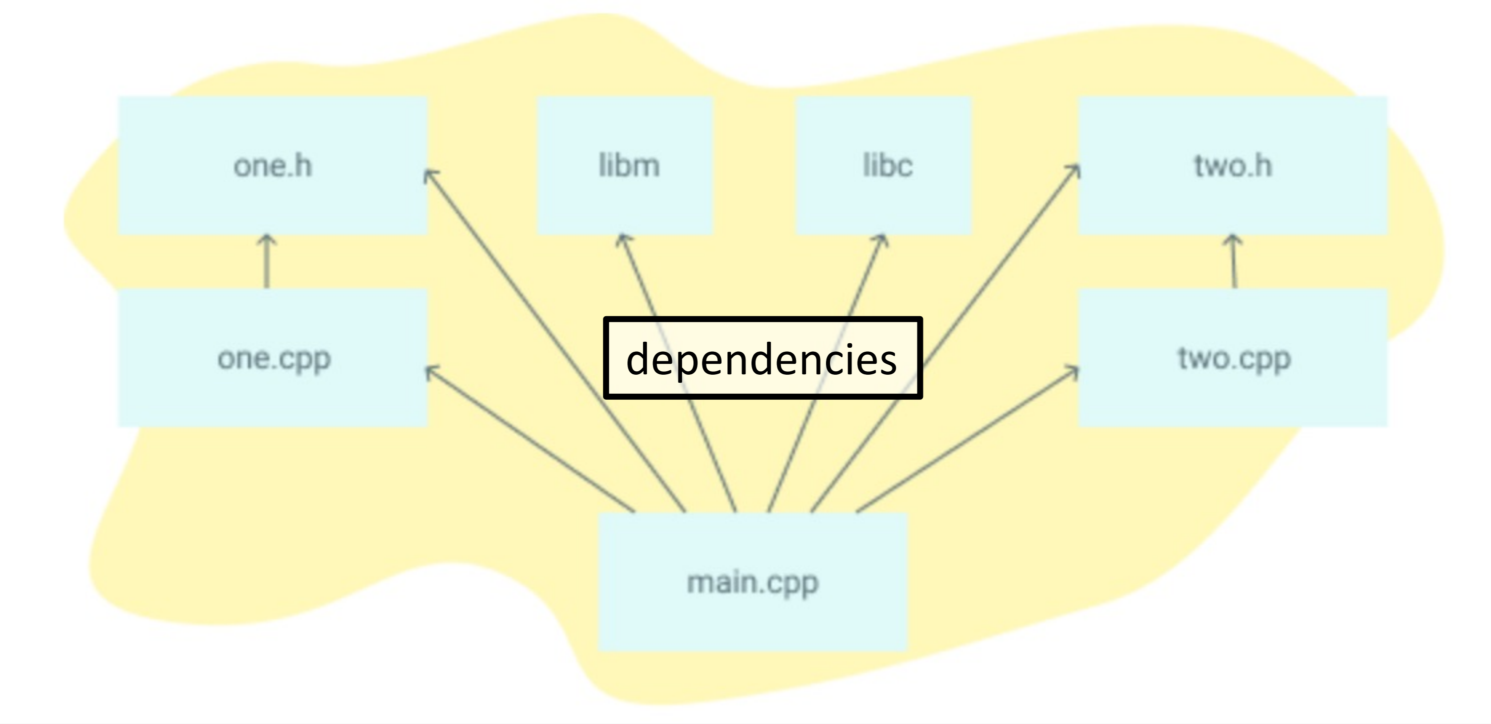
Building executables via g++

- Run the program g++ and tell it which cpp files you want built
  - In simple programs, it's just that simple
  - Options to include:
    - -g (leaves in debugging symbols)
    - -Wall (enables ALL warnings)
    - -o [filename] (tells g++ what to name the final program)
    - -std=c++11 (tells g++ to use the c++11 language standard)
- Could be more specific and build object files (*.o), then link those
  - Great for larger programs with LONG build times
  - Can actually do partial rebuilds based on which source files have changed

What is Make?

- A tool to help build software but platform-dependent
- Huge supply of documentation: https://www.gnu.org/software/make/manual/make.html
- Rely on a “makefile” to specify the compilation details
  - i.e., what are the source files, how to link them together
- **make [target] → make build → make run → make test**
- A tutorial: https://makefiletutorial.com/ (with examples)

What is **Make**?



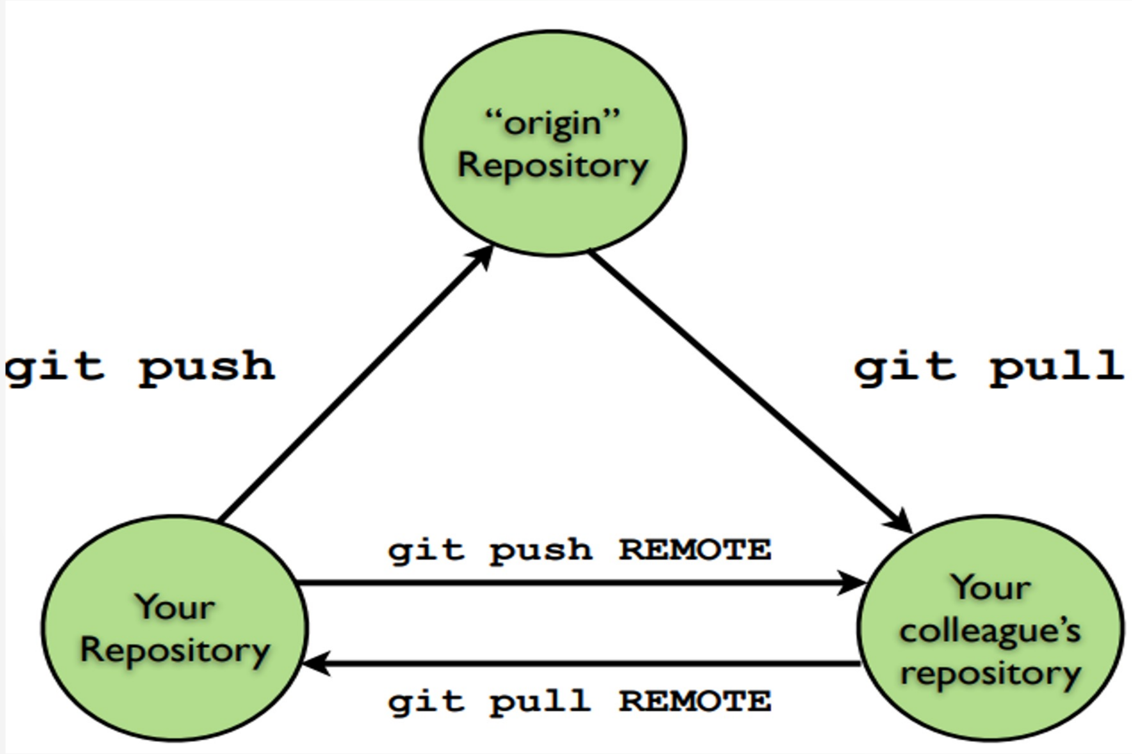
What is CMake?

- A **Cross-platform Make** tool using a compiler-independent and platform-independent method
- CMake is not a build system. It is a build-system generator
- CMake relies on a “**CMakeLists.txt**” file to generate makefile
- You can use the same C++ project in Linux, MacOS, Windows
  - With no or tiny modification
- You can also develop your project in IDE
- An example: https://github.com/DataOceanLab/CPTS-223-Examples **(try this later)**
- Installation: https://cmake.org/install/
- Comparison with make: https://prateekvjoshi.com/2014/02/01/cmake-vs-make/

What is a Git?

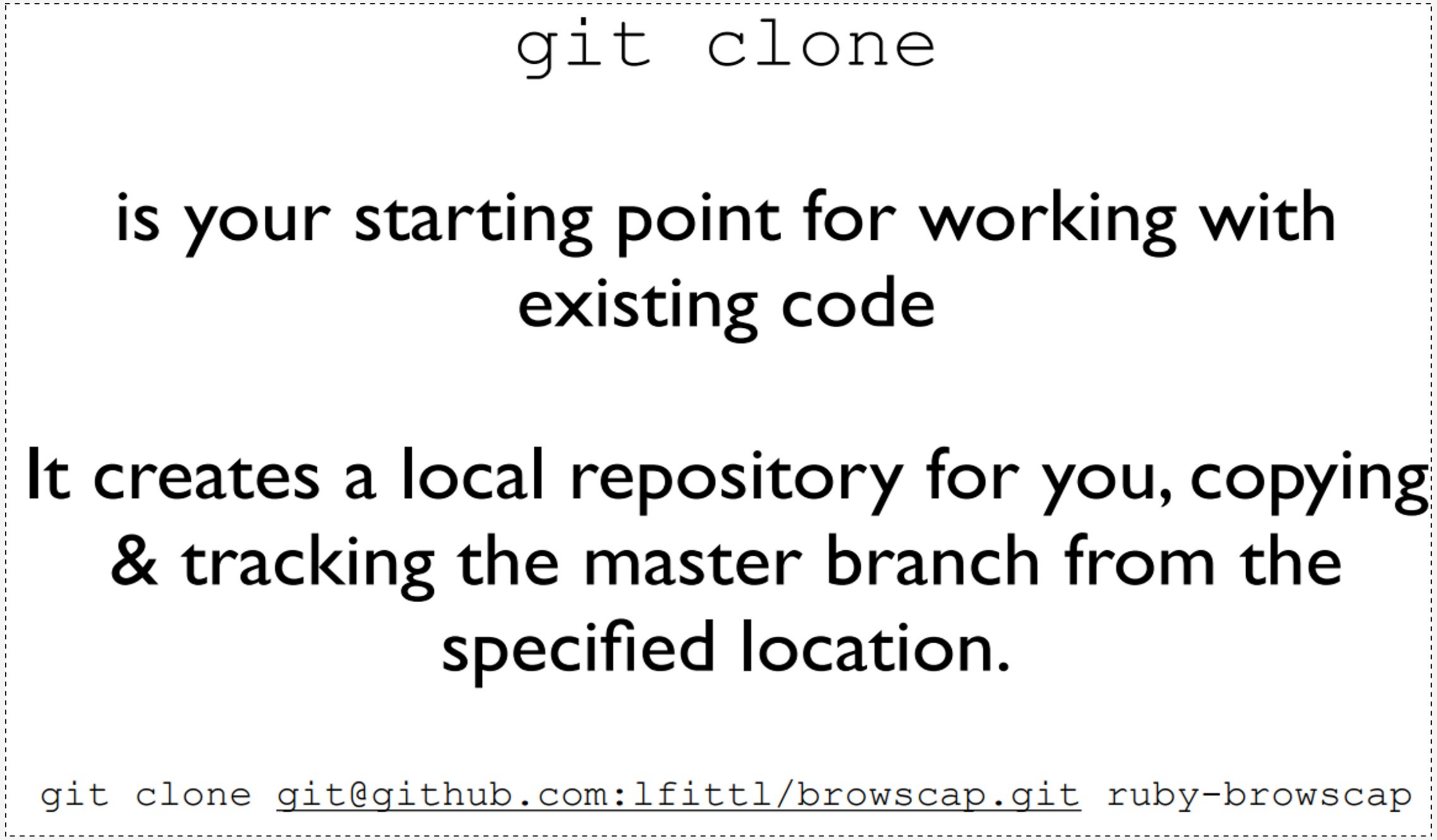
- A distributed version-control system for tracking changes in any set of files, originally designed for coordinating work among programmers
- Created by Linus for Linux kernel in 2005
- Install on Ubuntu: **sudo apt install git**
- SVN: a centralized version-control system. Good for big companies, requires a dedicated centralized server

What is a Git?



- You can push/pull commits to **any** remote repository, there is no real difference between a server and a client
- Distributed architecture

Copy a repo by cloning it



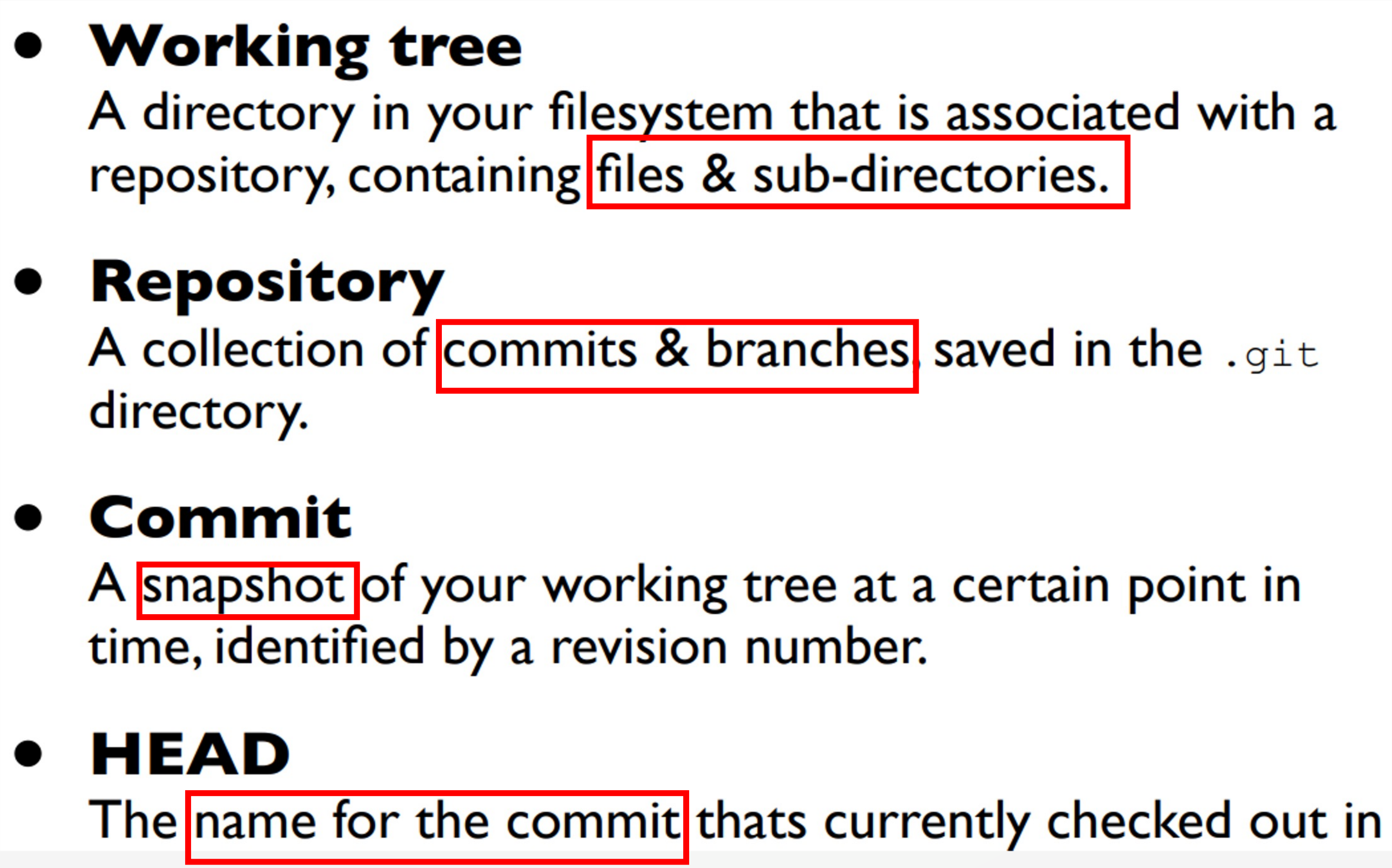
`git clone`

is your starting point for working with existing code

It creates a local repository for you, copying & tracking the master branch from the specified location.

```
git clone git@github.com:lfittl/browscap.git ruby-browscap
```

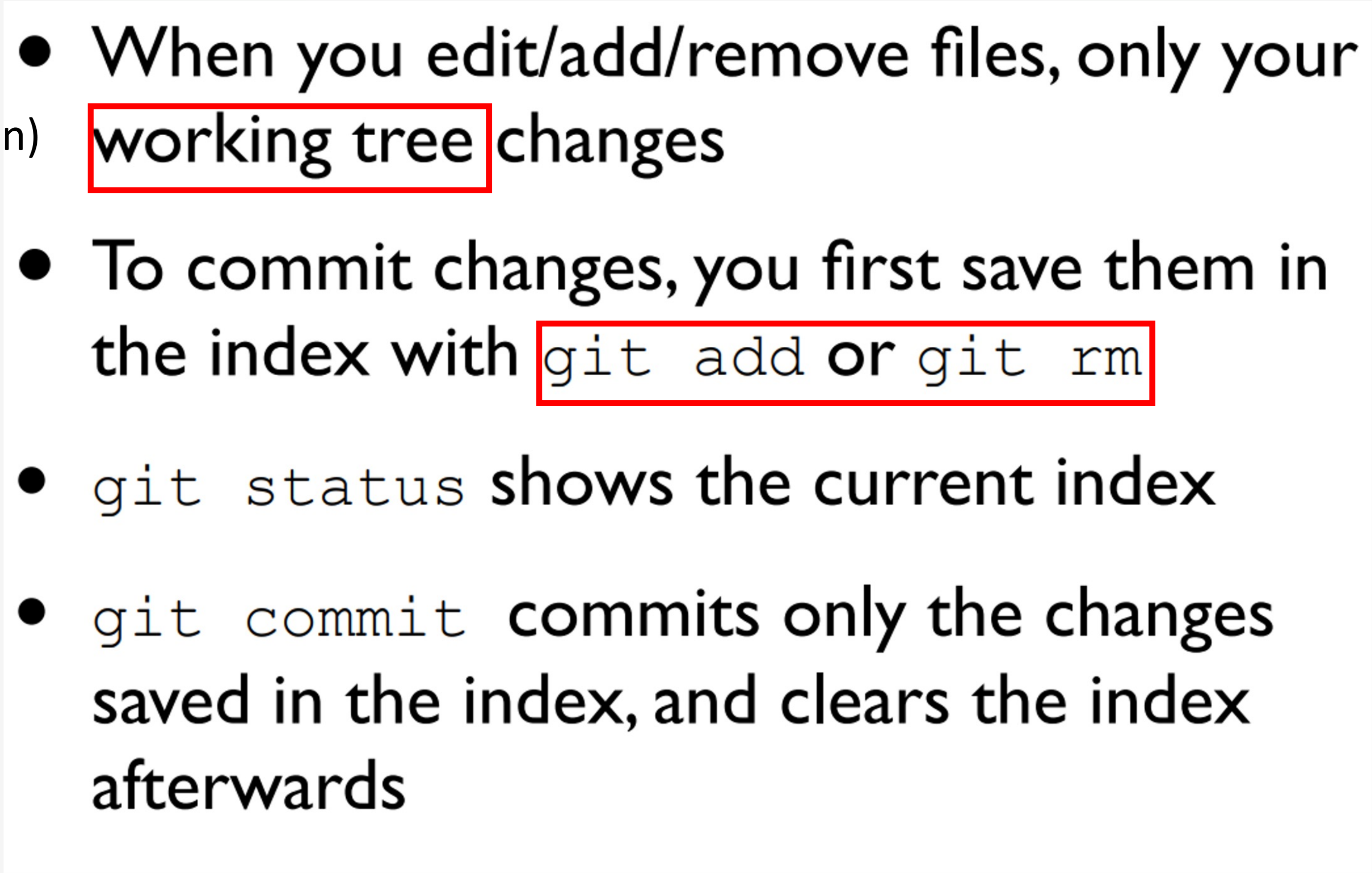
Concepts



- **Working tree**
  A directory in your filesystem that is associated with a repository, containing files & sub-directories.
- **Repository**
  A collection of commits & branches, saved in the `.git` directory.
- **Commit**
  A snapshot of your working tree at a certain point in time, identified by a revision number.
- **HEAD**
  The name for the commit thats currently checked out in

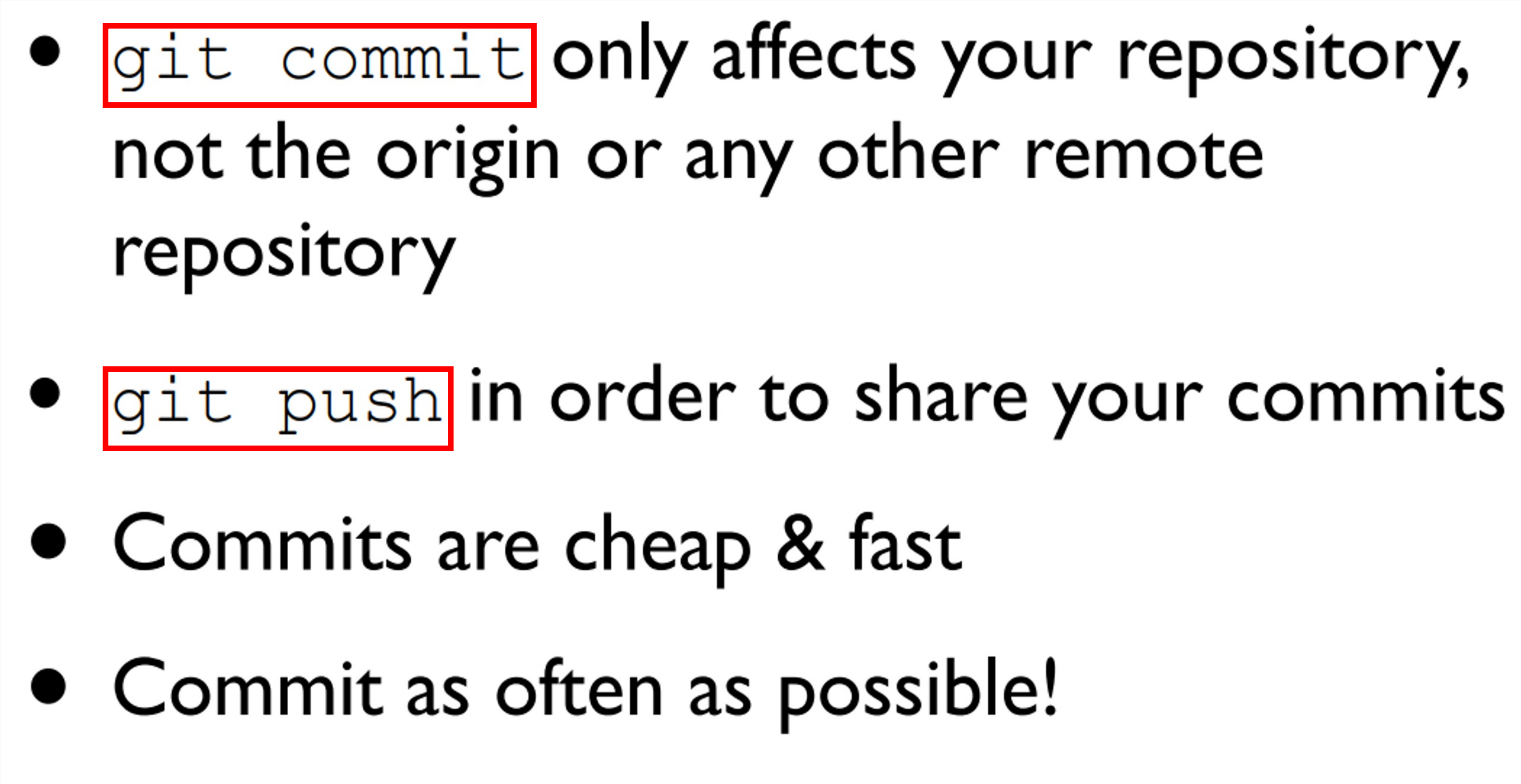
Staging changes for committing

(e.g., version)



- When you edit/add/remove files, only your working tree changes
- To commit changes, you first save them in the index with `git add` or `git rm`
- `git status` shows the current index
- `git commit` commits only the changes saved in the index, and clears the index afterwards

Commit and Push



- `git commit` only affects your repository, not the origin or any other remote repository
- `git push` in order to share your commits
- Commits are cheap & fast
- Commit as often as possible!

Version Control

GitHub

- Consider it a central place for a copy of your repo to live
- Web interface for management of things
- An example: https://github.com/DataOceanLab/CPTS-223-Examples **(try this now)**