Locality (fin) & git (start)

Intermediate Computer Systems Programming, Fall 2025

Instructor: Travis McGaha

TAs: Theodor Bulacovschi Ash Fujiyama

pollev.com/tqm

- How are you? Any feedback?

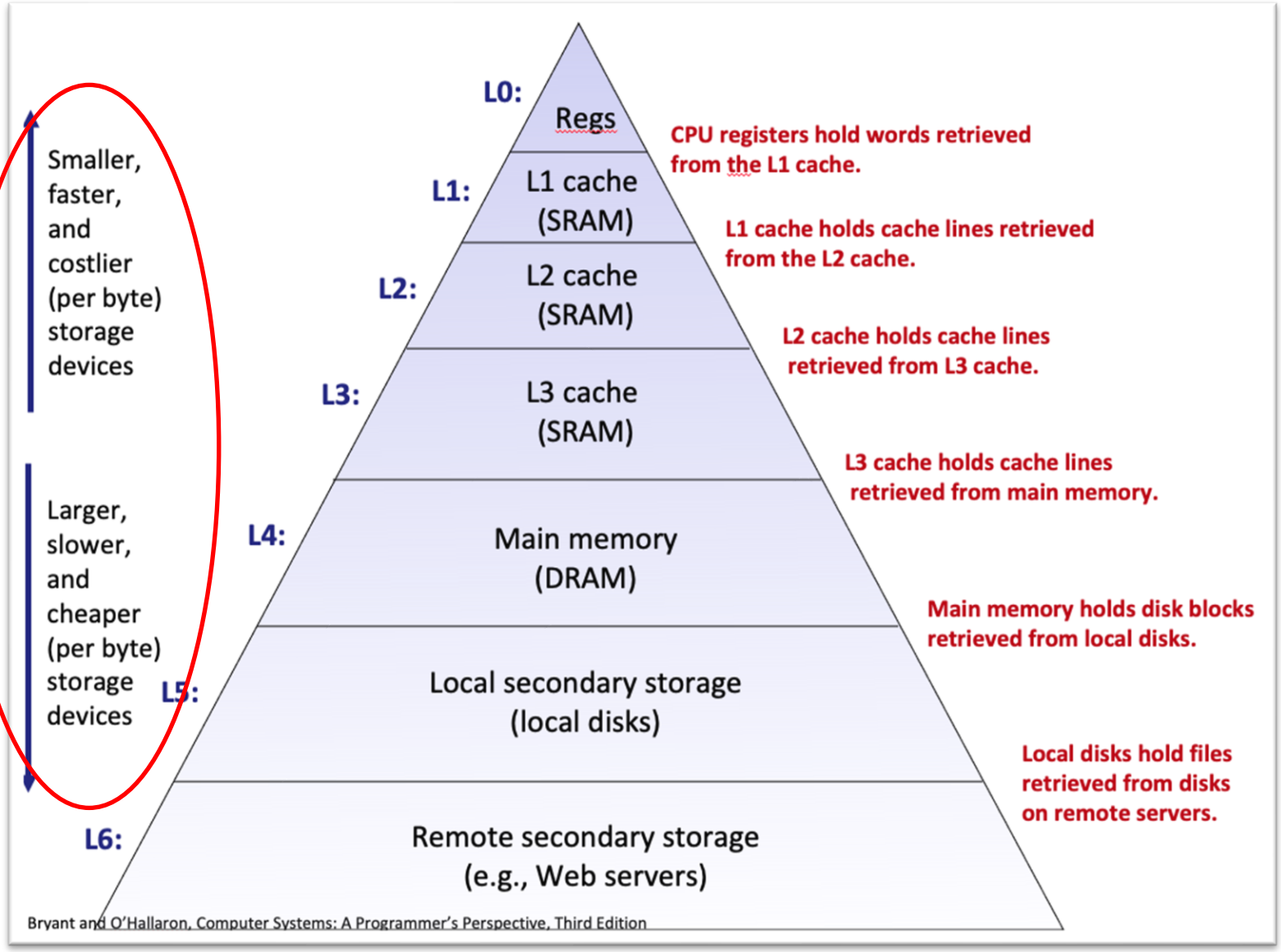
Administrivia

- HW04 Due Tomorrow
- HW05 posted Wednesday
  - Since you won't have everything you need till Wednesday
  - Should be a pretty short assignment, just a "hands on" for the git stuff so that you aren't as scared when you see it again :)
- Check-in Due before lecture
  - Assignment re-opens processed during/soon after lecture

Lecture Outline

- Locality Wrap-up
- git

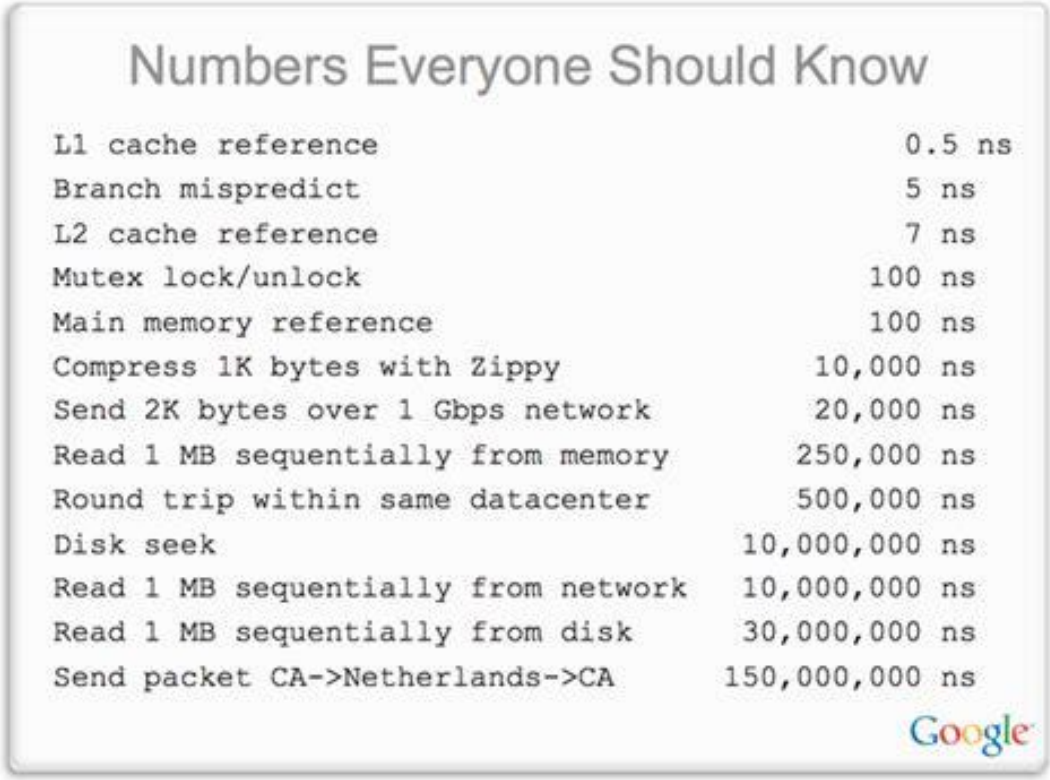
Memory Hierarchy (again)



Numbers Everyone Should Know

- There is a set of numbers that called “numbers everyone you should know”

- From Jeff Dean in 2009
- Numbers are out of date but the relative orders of magnitude are about the same



- More up to date numbers: https://colin-scott.github.io/personal_website/research/interactive_latency.html

Inheritance Refresher

- What does this print? (This is Java code )

```java
import java.util.ArrayList;
import java.util.List;

public class ICacheExample {
  public static void main(String[] args) {
    List<A> l = new ArrayList<A>();

    l.add(new A());
    l.add(new C());
    l.add(new A());
    l.add(new B());

    for (A item : l) {
      item.compute();
    }
  }
}
```

```java
public class A {
  public void compute() {
    System.out.println("A::compute");
  }
}

public class B extends A {
  public void compute() {
    System.out.println("B::compute");
  }
}

public class C extends A {
  public void compute() {
    System.out.println("C::compute");
  }
}
```

Instruction Cache

- The CPU not only has to fetch data, but it also fetches instructions. There is a separate cache for this
  - which is why you may see something like `L1I` cache and `L1D` cache, for Instructions and Data respectively

- Consider the previous example, it jumped between the different compute functions!

```
public class A {
  public void compute() {
    // …
  }
}
```

```
public class B extends A {
  public void compute() {
    // …
  }
}

public class C extends A {
  public void compute() {
    // …
  }
}
```

Instruction Cache

- Consider this code

```
public class ICacheExample {
  public static void main(String[] args) {
    List<A> l = new ArrayList<A>();
    // …
    for (A item : l) {
       item.compute();
    }
  }
}
```

```
public class A {
  public void compute() {
    // …
  }
}
```

```
public class B extends A {
  public void compute() {
    // …
  }
}

public class C extends A {
  public void compute() {
    // …
  }
}
```

- When we call item.compute that could invoke A's compute, B's compute or C's compute
- Constantly calling different functions, may not utilizes instruction cache well

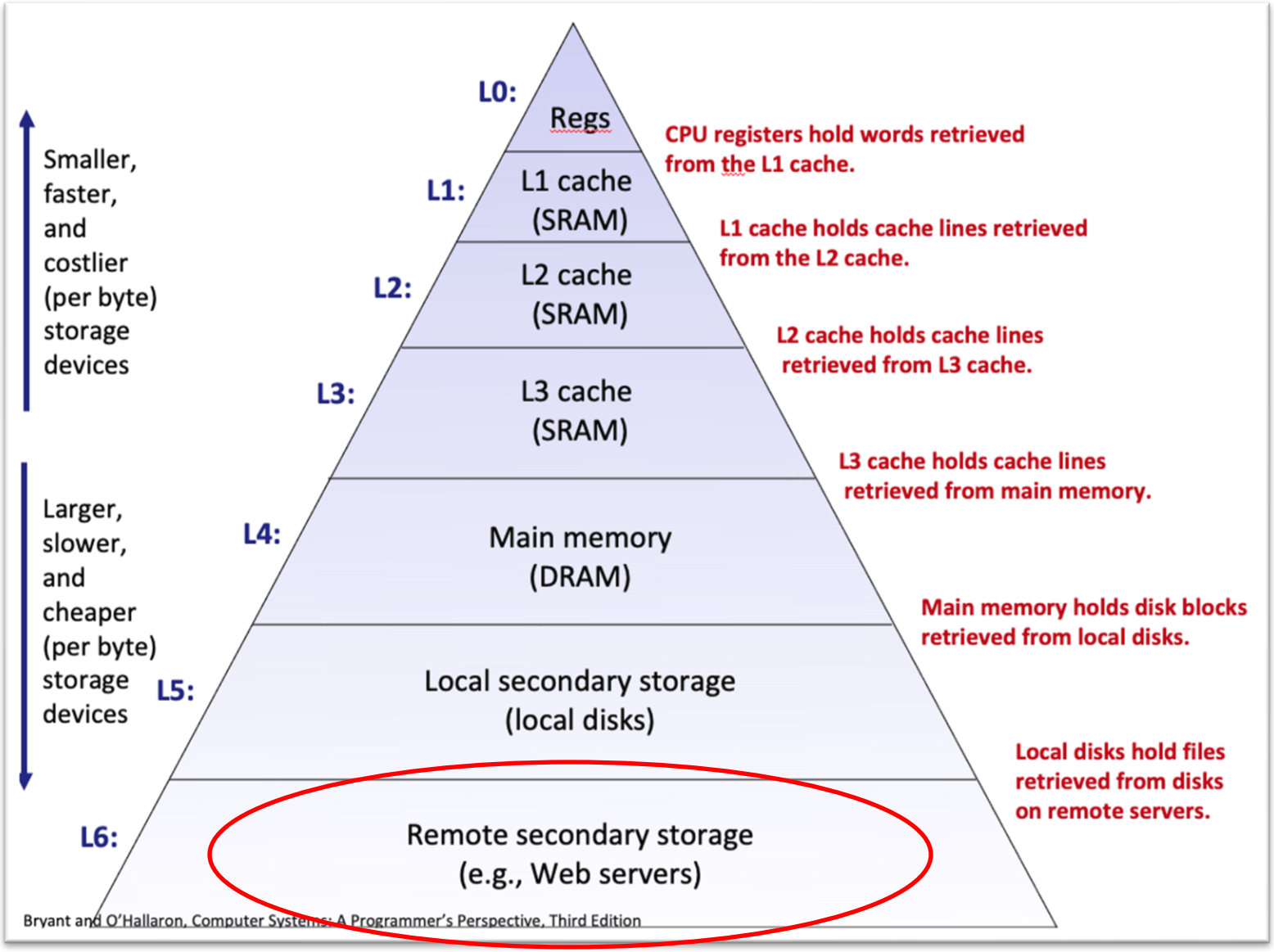
Instruction Cache

- Consider this code new code: makes it so we always do A.compute() -> B.compute() -> C.compute()

- Instruction Cache is happier with this ☺

```
public class ICacheExample {
  public static void main(String[] args) {
    List<A> la = new ArrayList<A>();
    List<B> lb = new ArrayList<B>();
    List<C> lc = new ArrayList<C>();
    // …
    for (A item : la) {
       item.compute();
    }
    for (B item : lb) {
       item.compute();
    }
    for (C item : lc) {
       item.compute();
    }
  }
}
```

Memory Hierarchy (again)



What is the network?

- We will talk about networking later, but for now:
  - When we say “networking” we mean one computer communicating with another computer (typically via the internet, but not necessarily).
  - Sending/Receiving data over the network at a low level ****looks**** very similar to reading/writing data to/from a file.

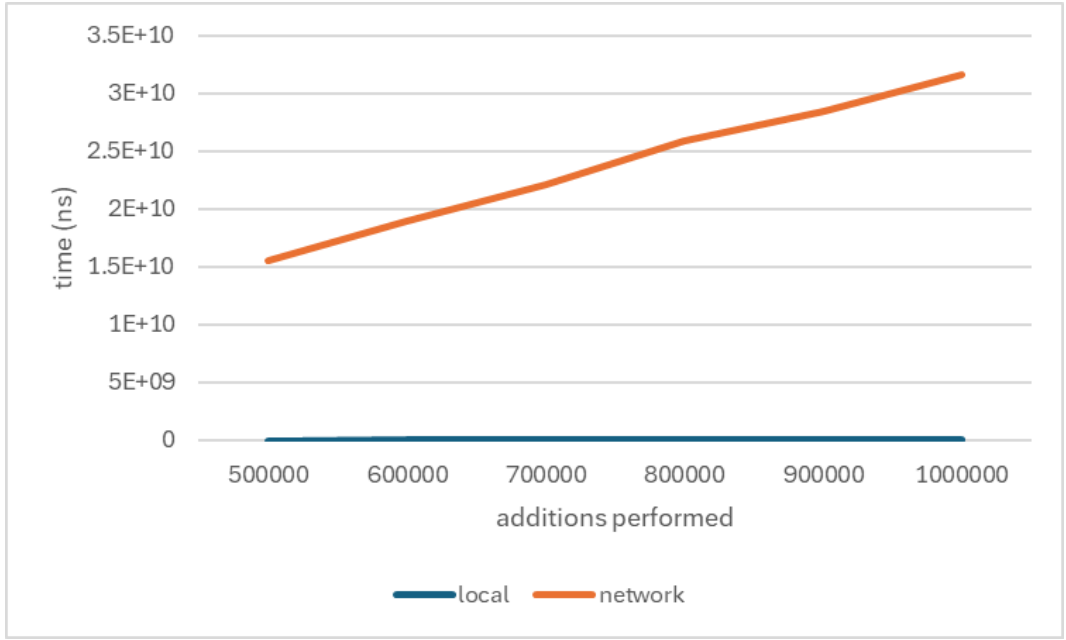
- Since there is a huge number of computers that could theoretically communicated with, the storage space is much larger than the storage space on our computer.

- Using the network has a huge overhead, even compared file Input/Output.
  - Can have benefits still (e.g. parallelism or scalability) that we will talk about later.

Crude (yet useful) example

- I wrote a program to add generate 1,000,000 pairs of 64-bit integers and then add each pair together (do 1,000,000 64-bit additions).
  - One program does the additions locally
  - One program generates the numbers, sends them over the network and then receives the result back.

- Which one is faster?
  By how much?



  - Local one is ~4,000 times faster.
  - This was even while the network test was the same computer connecting to a different program on the same computer. (If it was a different computer, it probably would be even slower)

TRAP: Network Drives!!!!!!!

- When navigating in file explorer/finder/the terminal there may be things that look like files/directories but are not actually “normal” files/directories.

- Some folders are actually “network drives”, meaning that the actual file contents are stored on a separate computer, and reading from the file involves reading it from the network.
  - This can be easily missed and can greatly affect the speed of a program.
  - Example: if you must feed terabytes of data from a file to a GPU to train an LLM.

Microservices

- Software architecture where you make a bunch of small and independent programs (services) that all communicate with each other over the network.
  - > network
  - If you aren't careful, this can lead to significant latency in performing a task
  - There are also many difficulties that come up with using the network that we may talk about later
- The network is still useful! But it is slow and complex. Be careful!

Efficiency

- What does it mean for software to be “Efficient”?

Efficiency

- What does it mean for software to be “Efficient”?

- Turns out, we are very vague when we talk about “efficiency” in computer science. Usually involves trying to keep some ratio low/high:
  - Performance: time/task or tasks/second ← what you probably thought of first
  - There is a lot more to code than just “performance”

Other Metrics:

- There is a lot more to code than just performance:
  - Correctness: right_outcome/program_execution (ideally 100%)
  - Privacy: data re-identifiability vs effort it requires
  - Security: minimize exploits
  - Accessibility: average time it takes a user to use your thing
  - Energy: watts/task
  - Memory: information / bytes
  - Readability: minimize time it takes to understand code
  - Maintainability: minimize the time it takes to debug/update code

Previous Example:

- What does this code do? Does it do it correctly?

```
void bar(const matrix<N, N>& m1, const matrix<N, N>& m2, matrix<N, N>& out) {
  for (size_t i = 0; i < m1.rows; i += r) {
    for (size_t j = 0; j < m2.cols; j += r) {
      for (size_t k = 0; k < N; k += r) {
        for (size_t ib = i; ib < i + r; ++ib) {
          for (size_t jb = j; jb < j + r; ++jb) {
            for (size_t kb = k; kb < k + r; ++kb) {
              out[ib, jb] += m1[ib, kb] * m2[kb, jb];
            }
          }
        }
      }
    }
  }
}
```

Previous Example:

- What does this code do? Does it do it correctly?

```
void foo(const matrix<N, N>& m1, const matrix<N, N>& m2, matrix<N, N>& out) {
  for (size_t i = 0; i < m1.rows; ++i) {

    for (size_t j = 0; j < m2.cols; ++j) {

      for (size_t k = 0; k < m2.rows; ++k) {
        out[i, j] += m1[i, k] * m2[k, j];
      }

    }

  }
}
```

Performance Analysis

- Both do matrix multiplication one on the left utilizes cache better.
  - (Times are in ns for multiplying two 1024 x 1024 matricies)
- Which one is better though? What does it mean to be better?

6,716,299 (ns)

```
void bar(const matrix<N, N>& m1,
         const matrix<N, N>& m2, matrix<N, N>& out) {
  for (size_t i = 0; i < m1.rows; i += r) {
    for (size_t j = 0; j < m2.cols; j += r) {
      for (size_t k = 0; k < N; k += r) {
        for (size_t ib = i; ib < i + r; ++ib) {
          for (size_t jb = j; jb < j + r; ++jb) {
            for (size_t kb = k; kb < k + r; ++kb) {
              out[ib, jb] += m1[ib, kb] * m2[kb, jb];
            }
          }
        }
      }
    }
  }
}
```

7,609,260 (ns)

```
void foo(const matrix<N, N>& m1,
         const matrix<N, N>& m2, matrix<N, N>& out) {
  for (size_t i = 0; i < m1.rows; ++i) {
    for (size_t j = 0; j < m2.cols; ++j) {
      for (size_t k = 0; k < m2.rows; ++k) {
        out[i, j] += m1[i, k] * m2[k, j];
      }
    }
  }
}
```

Memory Hierarchy

- Main Points:
  - It is more than just BigO analysis – The constants that are discarded in this analysis can matter A LOT.
  - There is more to your computer than just the CPU (Processor). Memory, file I/O, network, etc. can have big impacts on the performance of your code
  - These things are still useful, you should not completely abandon the network, file I/O, etc. But if your code is running slowly/needs to be faster, then don't forget to check things other than the CPU.
  - There is also more to code than just performance. Most of the time some concessions are made to make code simpler & easier to maintain code.

Lecture Outline

- Locality Wrap-up
- git

What point does git have?

- A few points may have come to mind:
  - Sharing code
  - Keeping a copy of code somewhere else
  - Saving progress

- These all can be summarized by “version control”
  - You know when you write a paper for a class (or you draft a resume) you may have multiple versions: final_paper_draft.doc, final_paperv1.doc, final_paperv3.doc
  - Or if you are using google sheets it keeps track of this history by noting down the edits made and you can go back to view previous versions.
    - Also keeps track of who made what changes
    - Stores a copy of the files remotely (in google drive) so it can be accessed via other computers

Git as version control

- Version control (via git) will allow us to:
  - Revert changes made to a file or the entire project
  - See what edits are made over time, by whom, and when

git != GitHub

- git and github are two separate entities
- git:
  - Originally created by Linus Torvalds for version control of the Linux Kernel
  - Free and open-source distributed software system
- GitHub:
  - A host owned by Microsoft.
  - Provides ways to control access, track bugs, request features, etc. for software projects
  - Uses git to handle the core version control
  - GitHub is not the only service like this, others like GitLab and GitBucket exist

Repository

- A repository, commonly referred to as a *repo*, is a location that stores a copy of all files, history, etc.
  - The working directory (or working tree) is different from the repository
  - When working via GitHub, your computer has a repository on it, and GitHub also maintains a separate copy of the repository

- Typically, you never want to put any results of compilation (e.g. executables or .o files) inside of the repo. Others can just rebuild those

Init/clone

- Can create a git repository based on the current directory you are in via

```
git init
```

- Used to clone / copy a repository that already exists somewhere (e.g. GitHub)

```
git clone <repo_location>
```

e.g.

```
git clone https://github.com/torvalds/linux.git
git clone git@github.com:torvalds/linux.git
```

- You can also create a repository via GitHub and then clone it to your device
- Repositories will have a hidden directory called `.git` that contains the core files needed by git.

Commits

- Git uses commits to track the different “versions” and edits made to a git repository

- A commit is sort of like a “checkpoint” or “snapshot” of a repository.
- Each commit contains information:
  - When it was created
  - Who created the commit
  - What was different between this commit and the previous commit
  - A unique hash ID

- After you modify/create a file, you add/stage it to be part of the next commit. Then you create the commit and optionally push
  - git add -> git commit -> git push

add

- Used to track a new file, or some updates to an existing file so that these changes are part of the next commit

- You probably used `git add .` Does anyone know what the dot means?
  - Adds all files in the current directory and any sub directories

- You can be more specific and only include some things
  - `git add file.txt`
  - `git add file.txt sub_directory/other_file.cpp`

- **`git stage`** does the same thing
- **`git reset <file>`** will unstage/unadd a file

.gitignore

- A **`.gitignore`** file specifies intentionally untracked files that Git should ignore using regex-like syntax patterns.
  - .gitignore is still just a file. You can create and edit it like you would a new .cpp file.
  - Since the file name starts with a `.` it is "hidden" by default.
    (**`ls`** will not show it unless do **`ls -a`**).

- Each line in a **`.gitignore`** file specifies a pattern for files to be tracked or not tracked

```
# a leading # indicates a comment. This is a comment!

*.pdf       # do not track any file that ends in .pdf and may have any

test_suite  # do not track any file named test_suite

/lib/       # do not track the lib directory or anything inside of it

!/lib/must  # Do track the /lib/must file! (combined with above, the rest of /lib is ingored)
```

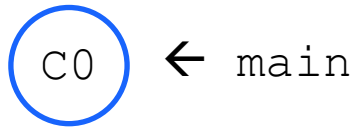
commit

- To create a commit based off of added/staged changes, we use `git commit`

- Each commit has:
  - When it was created, Who created the commit, What was different between this commit and the previous commit, A unique hash ID
    - Done for you based off of what you have already added, etc.
  - **A commit message!**
    - Which is like the "subject line" of an email describing the commit. You must supply this

- Use `git commit -m "your message here"` to create a commit with the specified message. If you don't provide a message like this, then it will prompt you in an editor you may not be familiar with (demo)

Version control as a tree of commits



- git manages our versions as a tree of commits.
- There can be multiple “branches” that diverge from each other.
  - For now, we will use one branch: “main”
  - Historically called “master”
  - A branch is a “history” of commits*
    (more on branches next lecture)

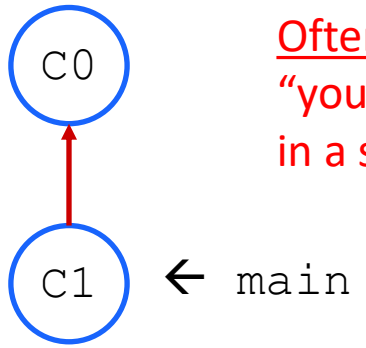
- Our repo usually starts with a commit:

Version control as a tree of commits

- git manages our versions as a tree of commits.
- There can be multiple "branches" that diverge from each other.
  - For now, we will use one branch: "main"
  - Historically called "master"
  - A branch is a "history" of commits*
    (more on branches next lecture)

- Our repo usually starts with a commit:
- Creating a new commit builds off of the previous (ancestor commits)

A branch refers to a specific commit.



Often, this is the "youngest" commit in a sequence.

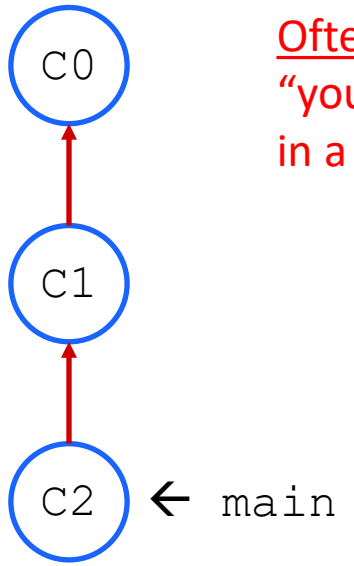
Version control as a tree of commits

- git manages our versions as a tree of commits.
- There can be multiple "branches" that diverge from each other.
  - For now, we will use one branch: "main"
  - Historically called "master"
  - A branch is a "history" of commits*
    (more on branches next lecture)

- Our repo usually starts with a commit:
- Creating a new commit builds off of the previous (ancestor commits)

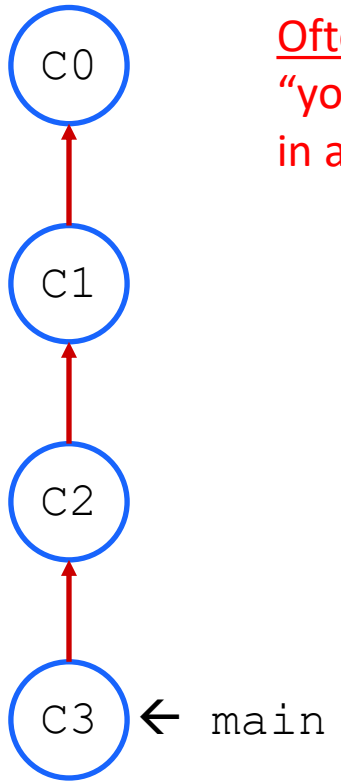
A branch refers to a specific commit.

Often, this is the "youngest" commit in a sequence.



Version control as a tree of commits

- git manages our versions as a tree of commits.
- There can be multiple "branches" that diverge from each other.
  - For now, we will use one branch: "main"
  - Historically called "master"
  - A branch is a "history" of commits*
    (more on branches next lecture)
- Our repo usually starts with a commit:
- Creating a new commit builds off of the previous (ancestor commits)



A branch refers to a specific commit.

Often, this is the "youngest" commit in a sequence.

Inspecting a repository

- `git log`: prints the commits made in your repository
  - Prints up to the current commit.you are on
  - This is similar to the diagram on the previous slide.
  - Each commit has their hash, commit message, ancestor commit, etc
  - `--graph` options to print it as a nice graph.

Inspecting a repository

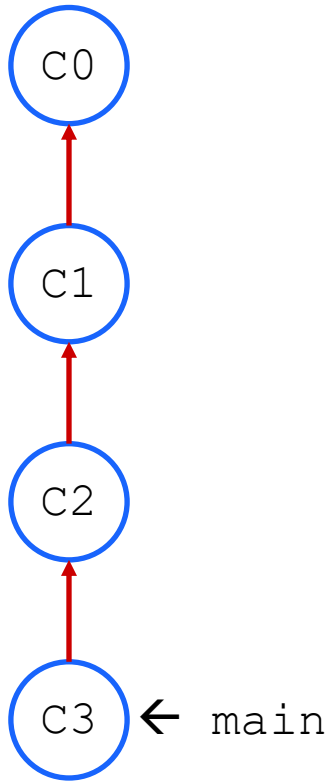
- `git status` will show which files are
  - Untracked by git
  - Modified but not staged for commit
  - Added to the staging area but committed.

- `git diff` can be used to compare the **diff**erence between commits/files
  - just `git diff` shows all changes/files tracked by git, not staged for commit
  - `git diff <commit hash>` will show the difference between your local repository and a local commit
  - Can specify a file name at the end to see differences on that file specifically e.g.
    - `git diff 30864f188a web/code/String.cpp`
    - `git diff web/code/String.cpp`

Git checkout & HEAD

- You can change your repo state to previous commits using `git checkout`

- `git checkout <commit_hash>`
  - Sets our repo to the specified commit
  - e.g. `git checkout c1`



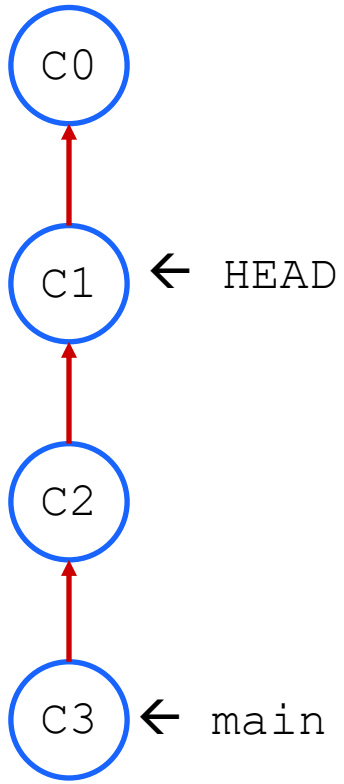
Git checkout & HEAD

- You can change your repo state to previous commits using `git checkout`

- `git checkout <commit_hash>`
  - Sets our repo to the specified commit
  - e.g. `git checkout c1`

- HEAD
  - A reference to the current branch/commit that you were on in your repository. Previous to now it was always the same as `main` so it was omitted.

- Changes made when looking in the past are easy to lose. Be careful!

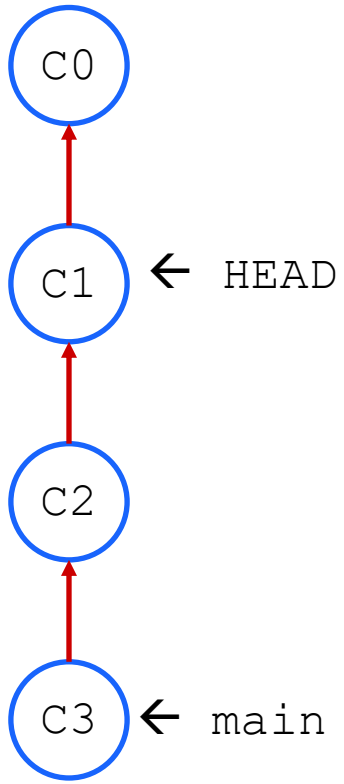


Git checkout & HEAD

- You can change your repo state to previous commits using `git checkout`

- `git checkout <commit_hash>`
  - Sets our repo to the specified commit
  - e.g. `git checkout c1`

- Here c1 is the commit hash. We can also checkout relative to HEAD or a branch (e.g. main)



Git checkout & HEAD

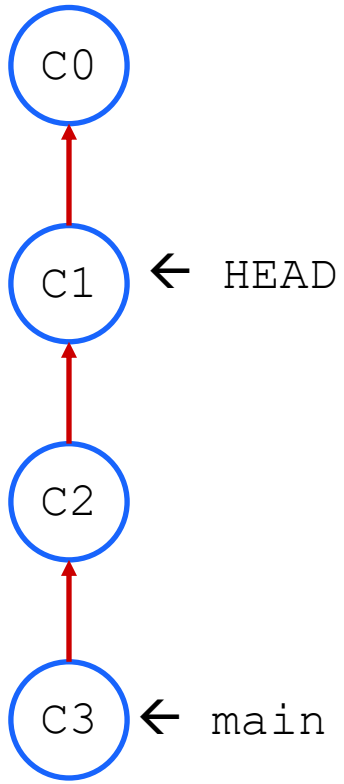
- You can change your repo state to previous commits using `git checkout`

- `git checkout <commit_hash>`
  - Sets our repo to the specified commit
  - e.g. `git checkout c1`

- Here c1 is the commit hash. We can also checkout relative to HEAD or a branch (e.g. main)

  `git checkout main^`

  Checks out the parent of main



Git checkout & HEAD

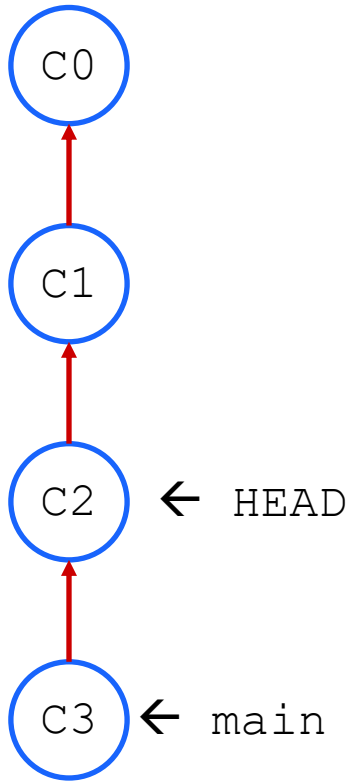
- You can change your repo state to previous commits using `git checkout`

- `git checkout <commit_hash>`
  - Sets our repo to the specified commit
  - e.g. `git checkout c1`

- Here c1 is the commit hash. We can also checkout relative to HEAD or a branch (e.g. main)

  `git checkout HEAD^^`

  Checks out the parent of the parent of head



Git checkout & HEAD

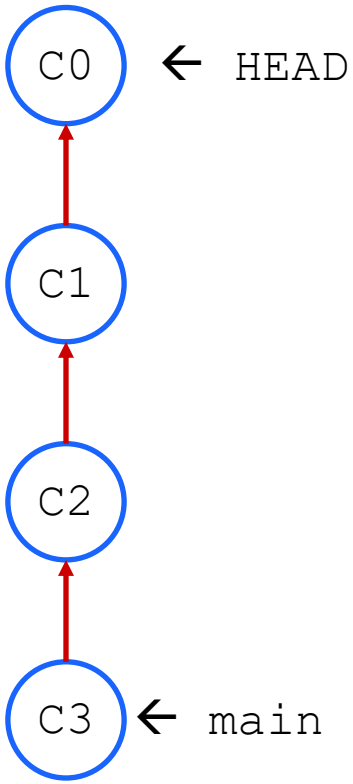
- You can change your repo state to previous commits using `git checkout`

- `git checkout <commit_hash>`
  - Sets our repo to the specified commit
  - e.g. `git checkout c1`

- Here c1 is the commit hash. We can also checkout relative to HEAD or a branch (e.g. main)

  `git checkout main`

  Go back to main

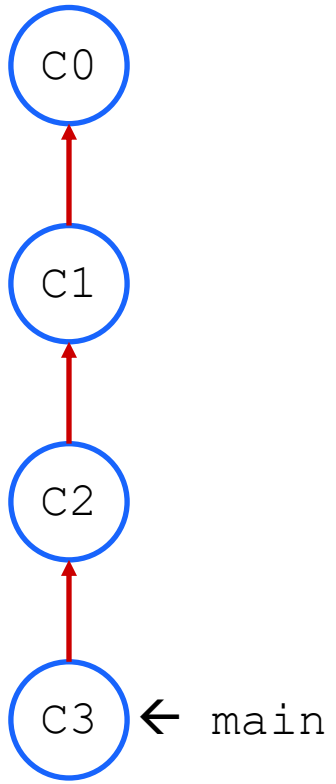


Git checkout & HEAD

- You can also use checkout to reset a file to the state it had in a previous commit:

- `git checkout -- dir/file.txt`
  - Resets `dir/file.txt` to the version from most recent commit

- `git checkout <hash> -- dir/file.txt`
  - To go to a specific commit



Commit Hash

- When you see commit hashes (e.g. through git log), you will see that they are really long:
  - E.g. 30864f188a7ba385d1de423c79242782d5daf964

- Most of the time, only the first 6 to 8 characters are needed to identify a commit

- `git checkout 30864f188a7ba385d1de423c79242782d5daf964`
  - Full commit hash
- `git checkout 30864f18`
  - First 8 characters

Some Poll Questions

- See Ed ☺

Remote

- Notice how we left off push and pull till now!
- Remember git != github, you can use git without having a “remote” repository.

- Remote repository: versions of your repository that are hosted elsewhere (e.g. in github or on a different location on your computer)

- Pull, push, merge, etc are used to synchronize your local repository with any remote repositories

- Can type `git remote show` to list the name of remote repos
  - `git remote show <name>` to see details about a specific connection

Push & Pull

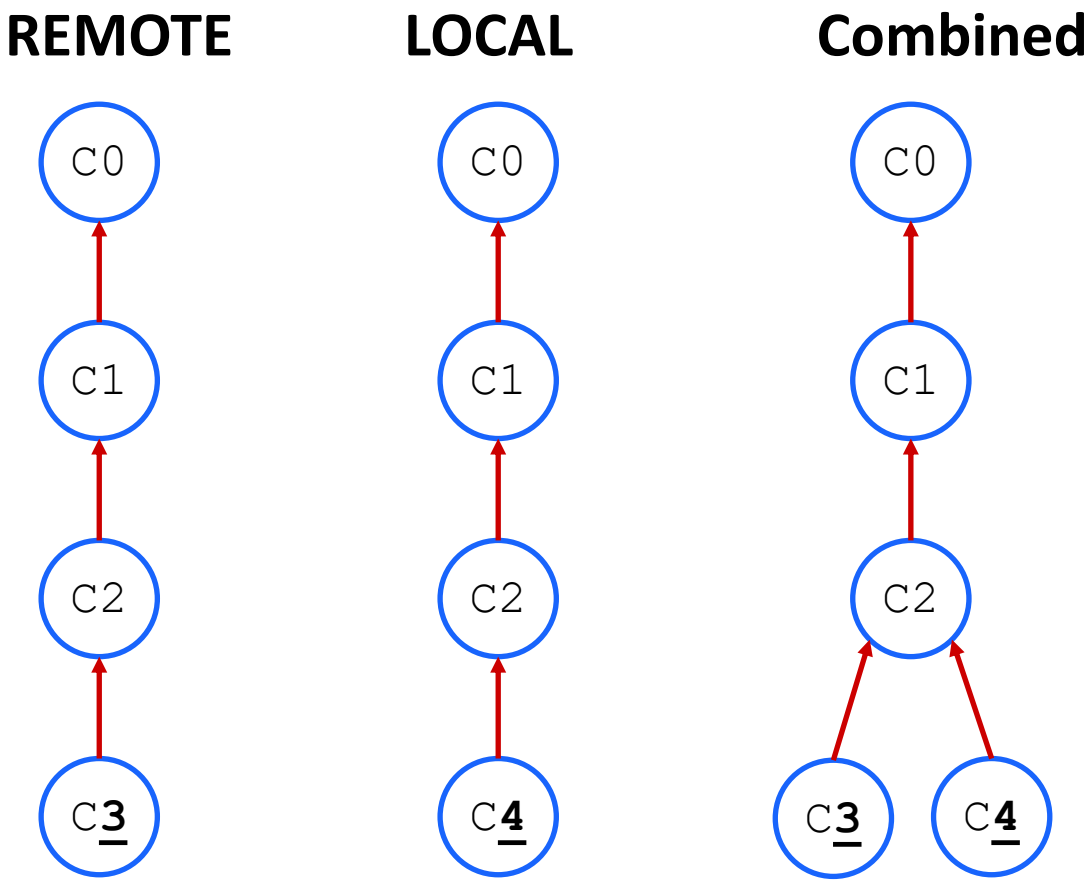
- `git push`
  - updates the remote repository with any commits you have locally

- `git pull`
  - updates local repository with any commits that are in remote, but not local

- Can encounter an error if multiple pushes cause a “fork” in the commit tree…
  - Usually cause a teammate pushed to the same branch before you.

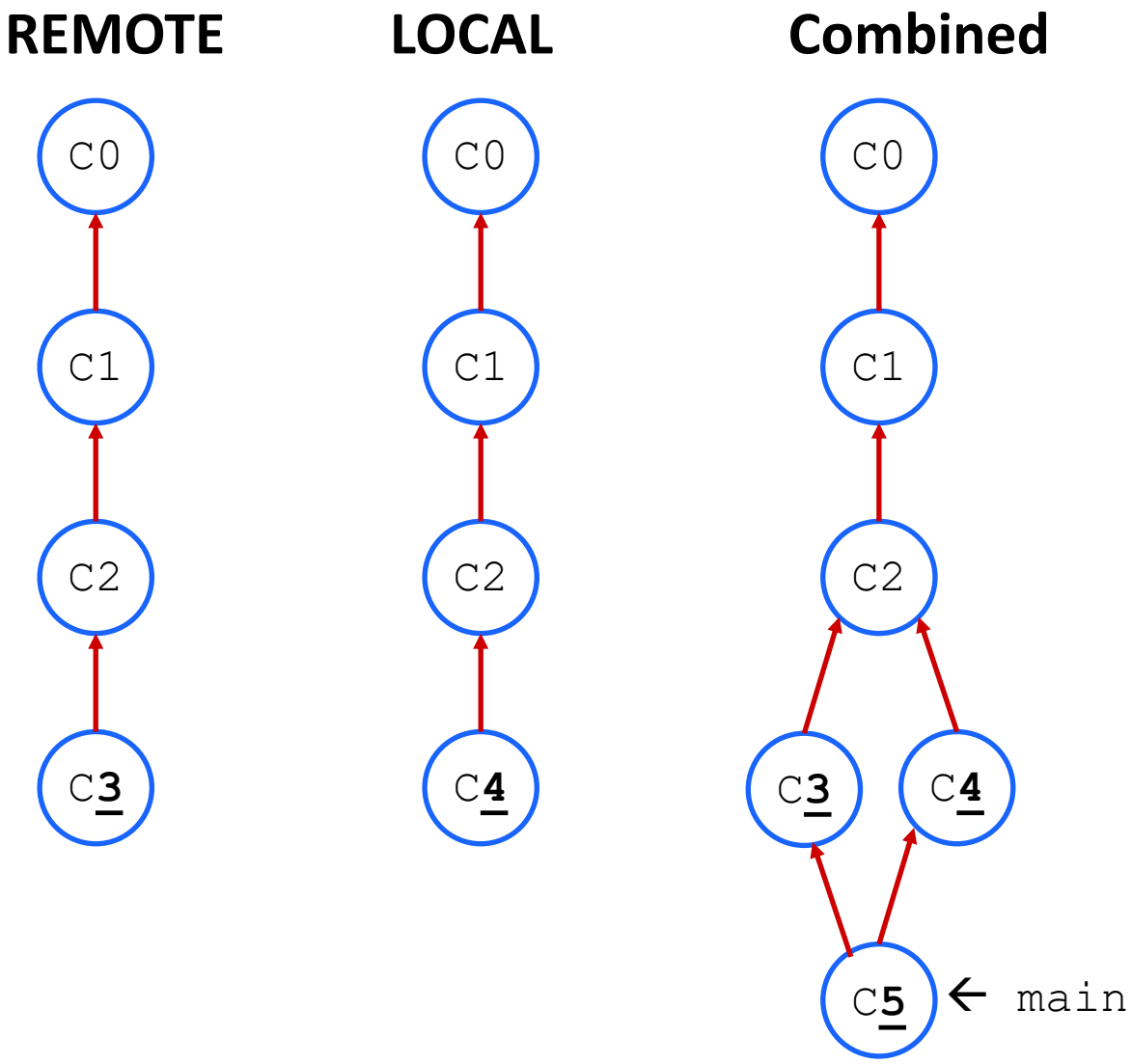
Merge

- **`git merge`** integrates a remote repo with local repo
- Sometimes there are conflicts:
- Should main refer to C3 or C4? They both have C2 as an ancestor…



Merge Conflict

- One solution: Combine the two into one commit that has changes from both
- `git merge` can do this automatically if there is no overlap in the differences between the two.
  - (they change different files, or different parts of files)



Merge Conflict (Manual intervention)

- If the changes are too close to each other/overlap, then manual intervention is necessary to decide what to keep / what to change.
- Lets say we had the code: `cout << "I am" << endl;`
- We change the code to: `cout << "I am a goofy goober" << endl;`, and push it
- But just before us, a colleague pushed: `cout << "I am a weirdo" << endl;`

- Git will tell us our pushed failed, and we must pull and merge changes. Automatic merge will fail and our file will update to look something like:

```
<<<<<<< HEAD (Current Change)
  cout << "I am a goofy goober" << endl;
=======
  cout << "I am a weirdo" << endl;
>>>>>>> 30864f18 (Incoming Change)
```

Merge Conflict (Manual intervention)

- Git will tell us our pushed failed, and we must pull and merge changes. Automatic merge will fail and our file will update to look something like:

```
<<<<<<< HEAD (Current Change)
  cout << "I am a goofy goober" << endl;
=======
  cout << "I am a weirdo" << endl;
>>>>>>> 30864f18 (Incoming Change)
```

- This is just characters in a file. Nothing special here. The characters just mark what our current change was, and what the remote change was .
- To resolve, it, delete the <<<<<<<, =======, >>>>>>> lines and change the code to whatever you want it to be. This could be one of the two changes given, or a combination:

```
cout << "I am a weird goofy goober" << endl;
```

  - Then add, commit, push etc.

That's all for now!

- Next time:
  - Git wrapup
  - Processes & The OS some more ☺

- Hopefully you are doing well ☺