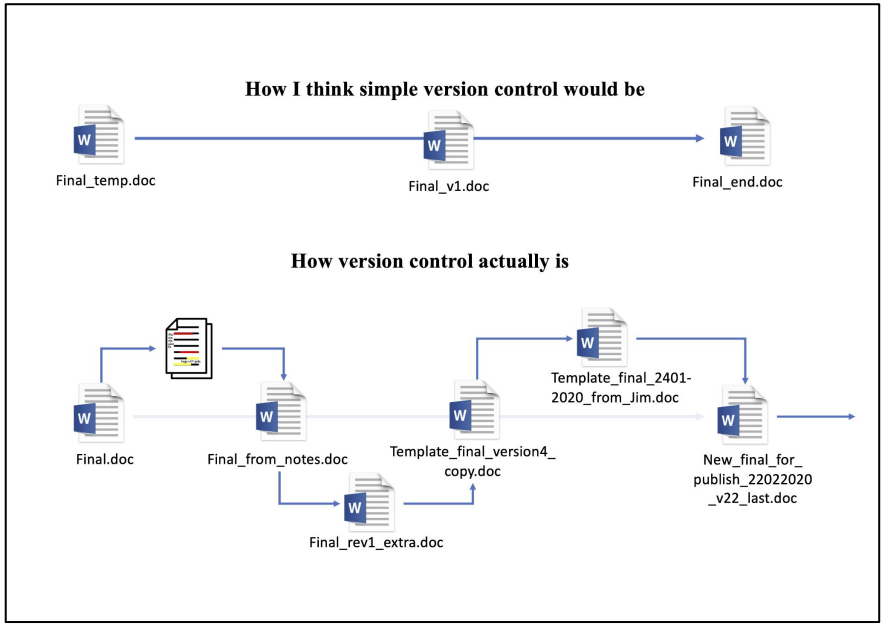
CSE 374 Programming Concepts and Tools

Lecture 06 - Version Control, Git

Today

We'll learn about:

- Version Control
- `git`



Discussion

Turn and talk:

Share a sad, funny, or frustrating story about a time you lost or messed up a version of your work (could be homework, a personal project, etc.) —what happened, and what did you learn from it?

Version Control

User Story: Individual

Does any of the following sound familiar?

- Your code was working great! Then *you made a few changes* and now everything is broken and you **saved over the previous version**?
- You **accidently deleted** a critical file and can't get it back.
- Your computer was broken or stolen and now **all of your files are gone**!
- While writing a paper for one of your classes you save each version as **final_paper.doc**, **final_paper2.doc**, **final_paper_actually_this_time.doc**, **UGH.doc**

There has to be a better way to manage versions...

User Story: Team

Does any of the following sound familiar?

- My partner and I are paired up for a project for one of our classes. We usually work together in-person, but sometimes we have to work remotely.
  - **Who keeps the most up-to-date version of the project?**
  - How do we **share changes** with each other?
  - What if I want to compare the changes my partner made?
- Someone changed or removed all my work, so now I have to do it all over again.
- How do we **keep backups** of important files? Who stores them on their computer?

Version Control

Version Control: Software that keeps track of changes to a set of files.

You likely use version control all the time:

- `Ctrl+Z` to undo changes, `Ctrl+Y` to redo them
- In Google Docs you can see who made what changes to a file

There are many use-cases for version control!

- Musicians tracking different versions of songs
- Video or movie editors
- Law firms tracking document changes over time

Examples: subversion, perforce, mercurial, cvs, sourcesafe, **git**

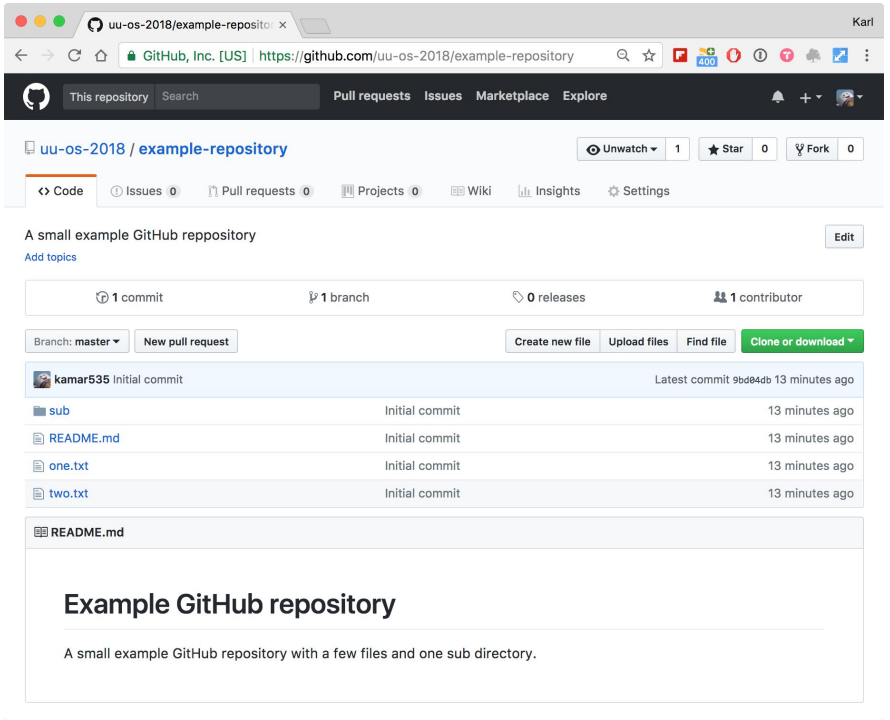
Git

Repository

A repository, commonly referred to as a repo, is a location that stores a copy of all files.

Analogous to the root directory in a file system (i.e. '/').

- The "project root"



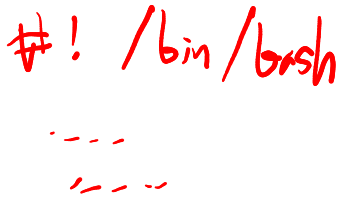
Repository Do's and Don'ts

What should be stored inside of a repository?

- Source code files (*i.e.*, `.c` files, `.java` files, etc.)
- Build files (Makefile)
- Documentation files

What should NOT be stored inside of a repository (generally)

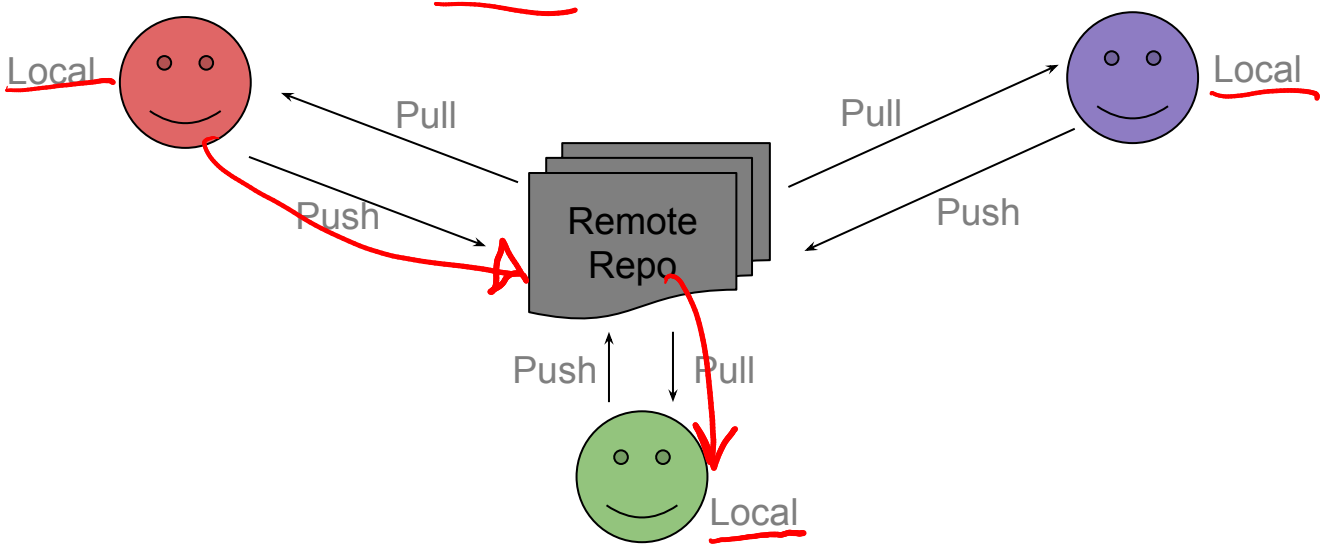
- Generated files
  - Object files (i.e. `.class` files, `.o` files)
- Executable binary files (executable scripts are OK!)



More on these types of files when we start with C!

Sharing Changes

- With git, everyone working on the project has a **local** copy of the repository and its history
  - The local copy is where we make our own changes.
  - We share changes by *pushing* and *pulling*



Central Git Repository

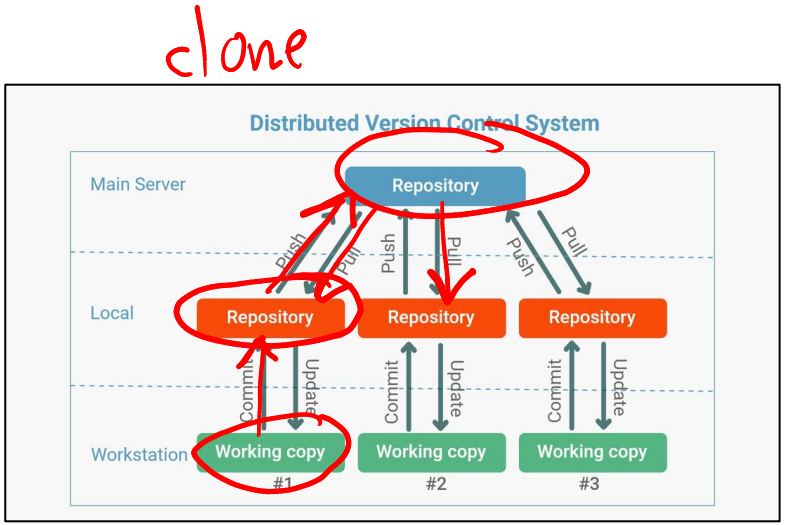
Keep an "origin" copy of the repo on a Git server

- The remote repository is the *defacto* central repository

Each user ***clones*** the repo to create a local copy

A user makes changes, ***add*** s and ***commit*** s those to their copy and ***push*** to save them in the remote repository

All users ***pull*** from the central server periodically to get changes (instead of from



Git: Four Phases

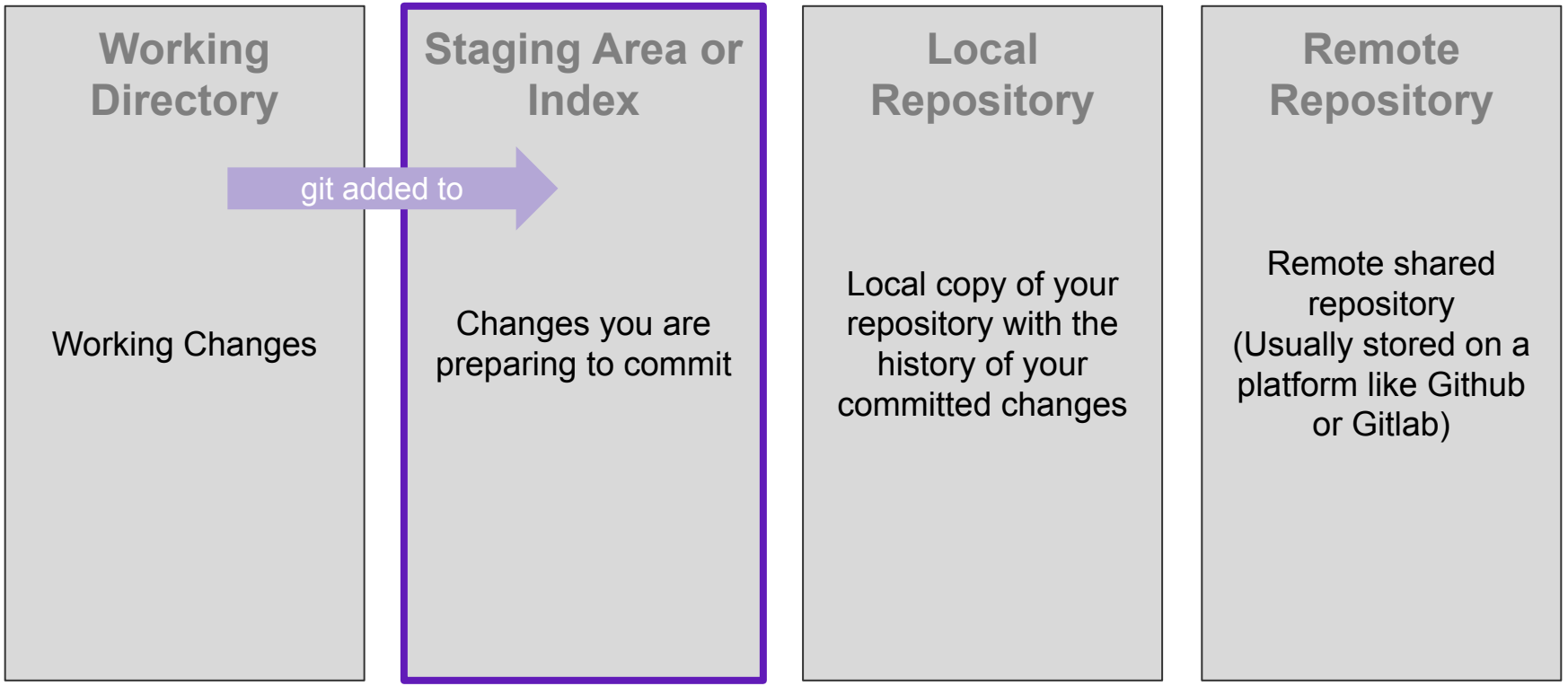
ls

Working Directory	Staging Area or Index	Local Repository	Remote Repository
Working Changes	Changes you are preparing to commit	Local copy of your repository with the history of your committed changes	Remote shared repository (Usually stored on a platform like Github or Gitlab)

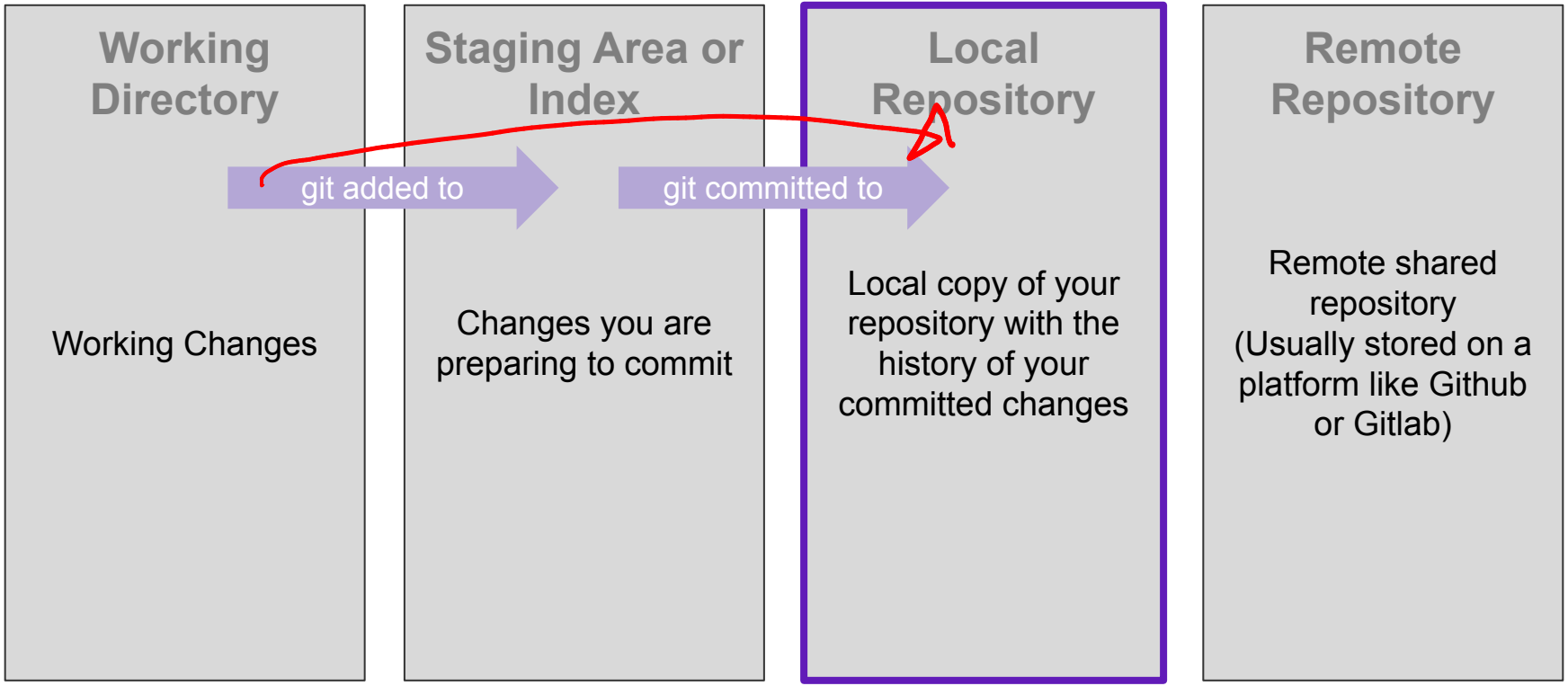
Git: Four Phases

Working Directory	Staging Area or Index	Local Repository	Remote Repository
Working Changes	Changes you are preparing to commit	Local copy of your repository with the history of your committed changes	Remote shared repository (Usually stored on a platform like Github or Gitlab)

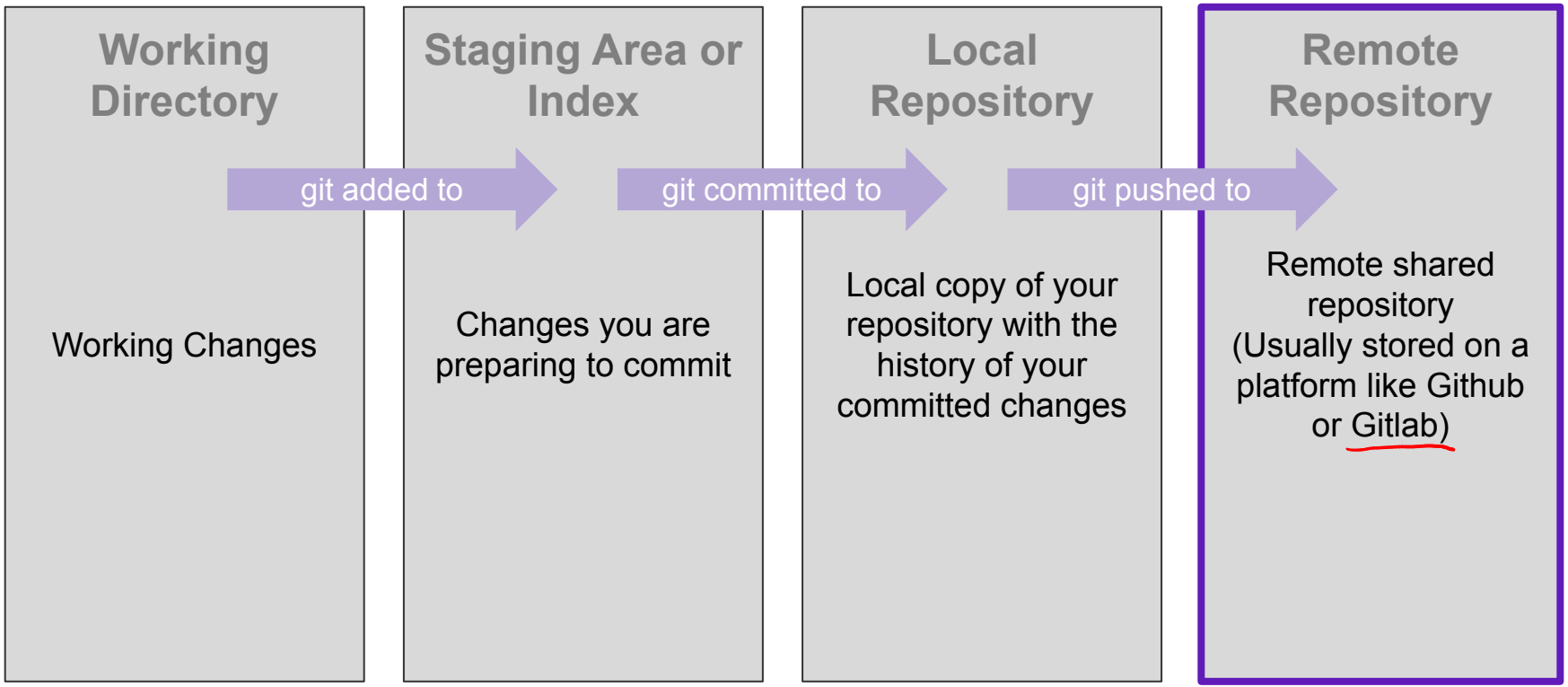
Git: Four Phases



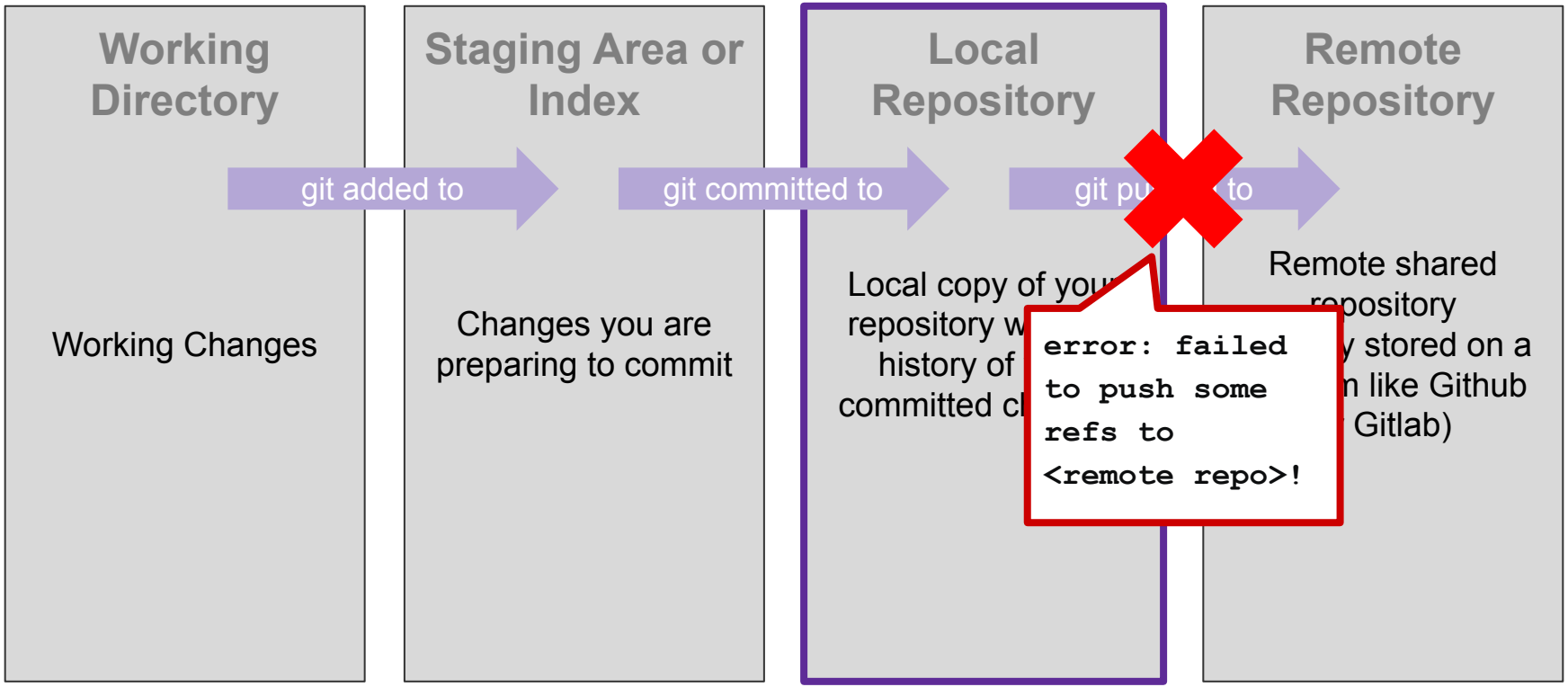
Git: Four Phases



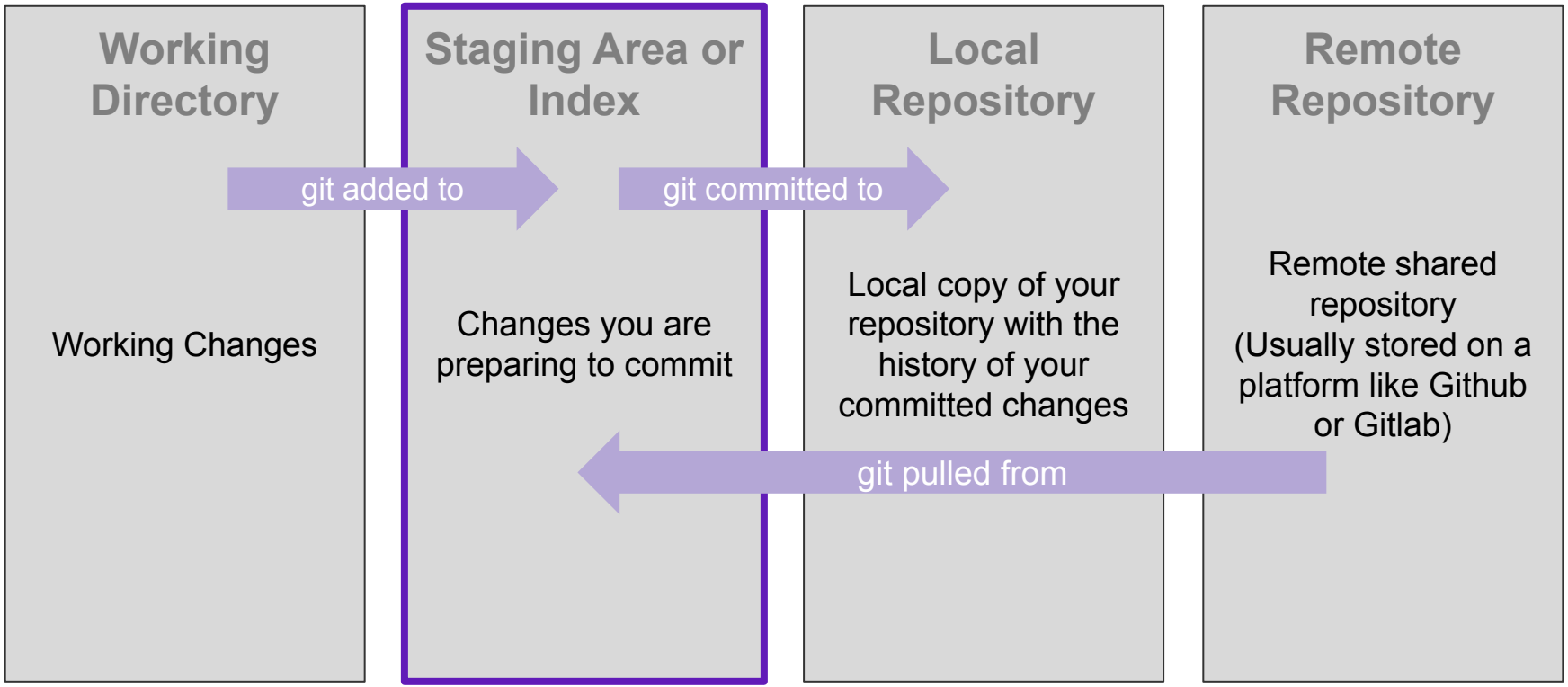
Git: Four Phases



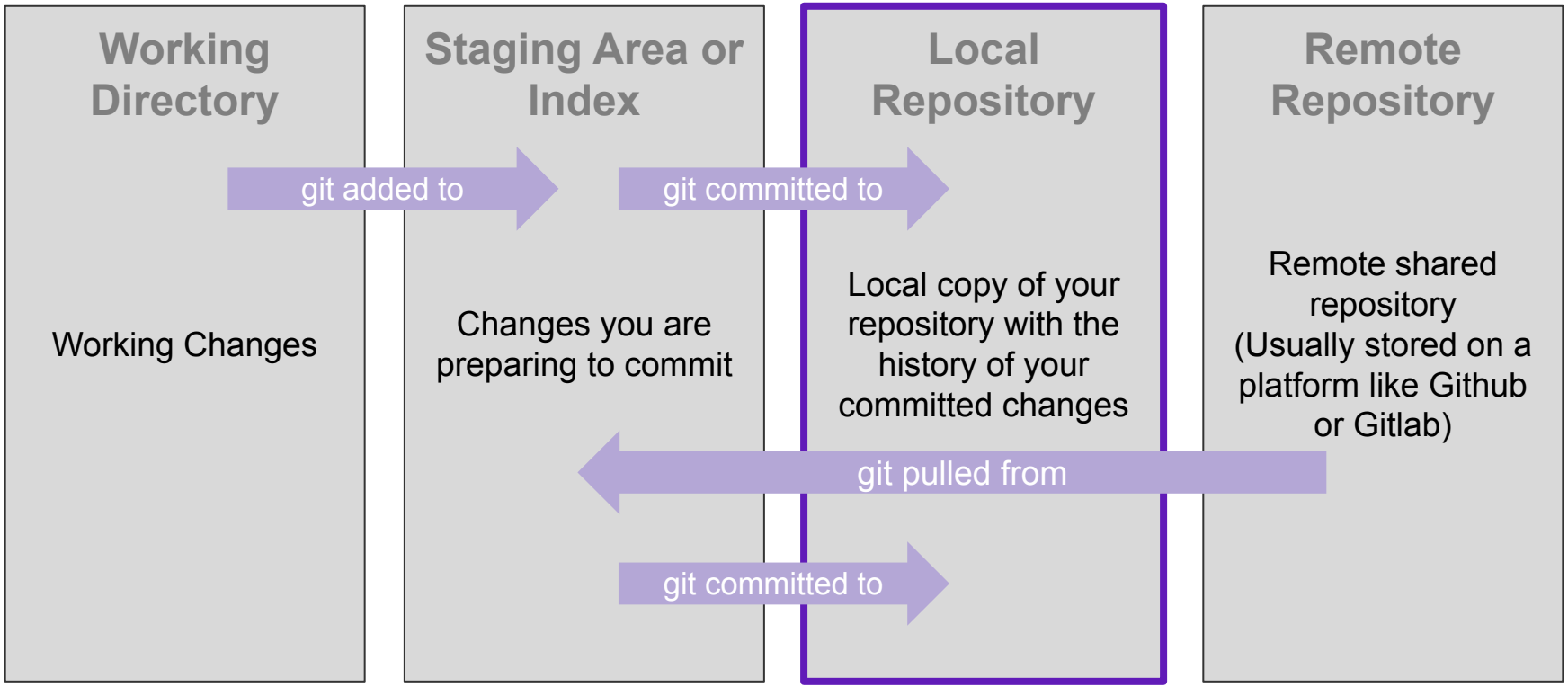
Git: Four Phases



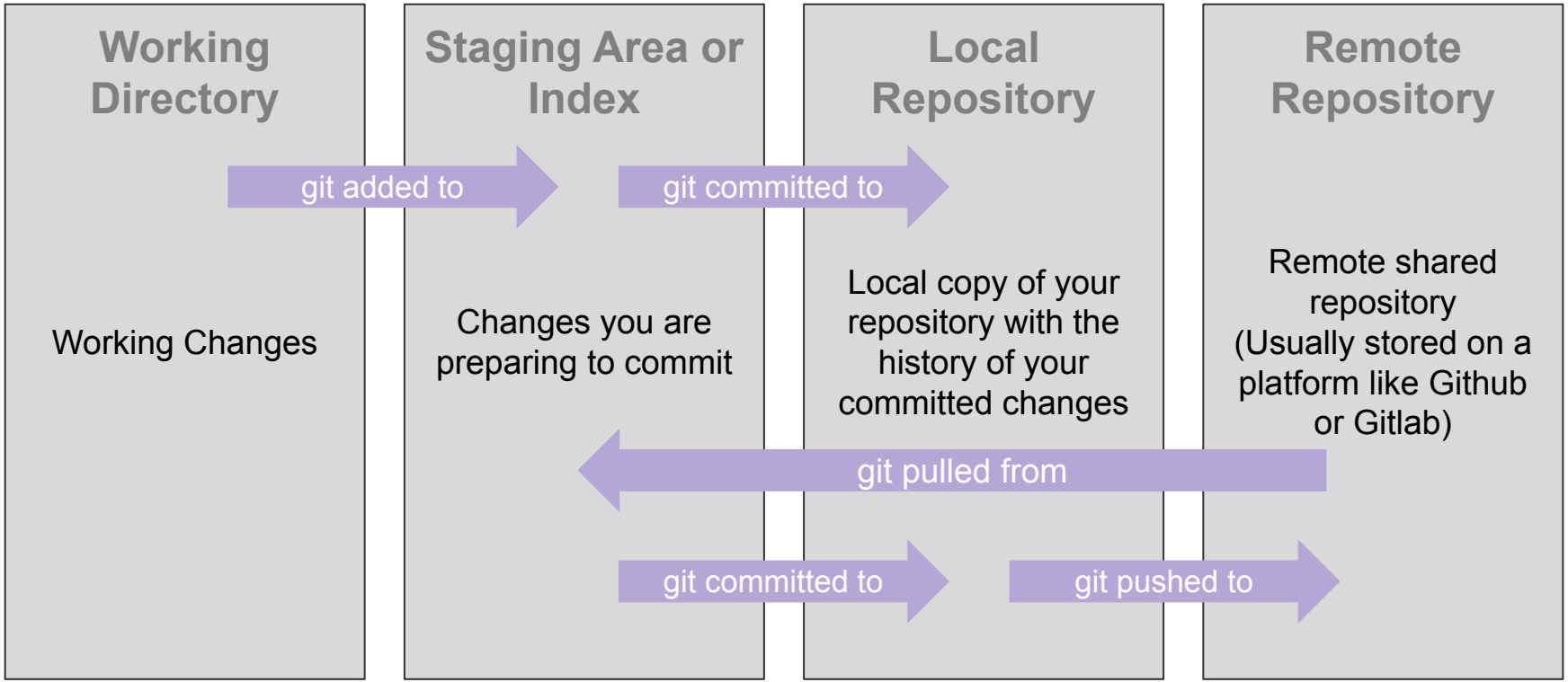
Git: Four Phases



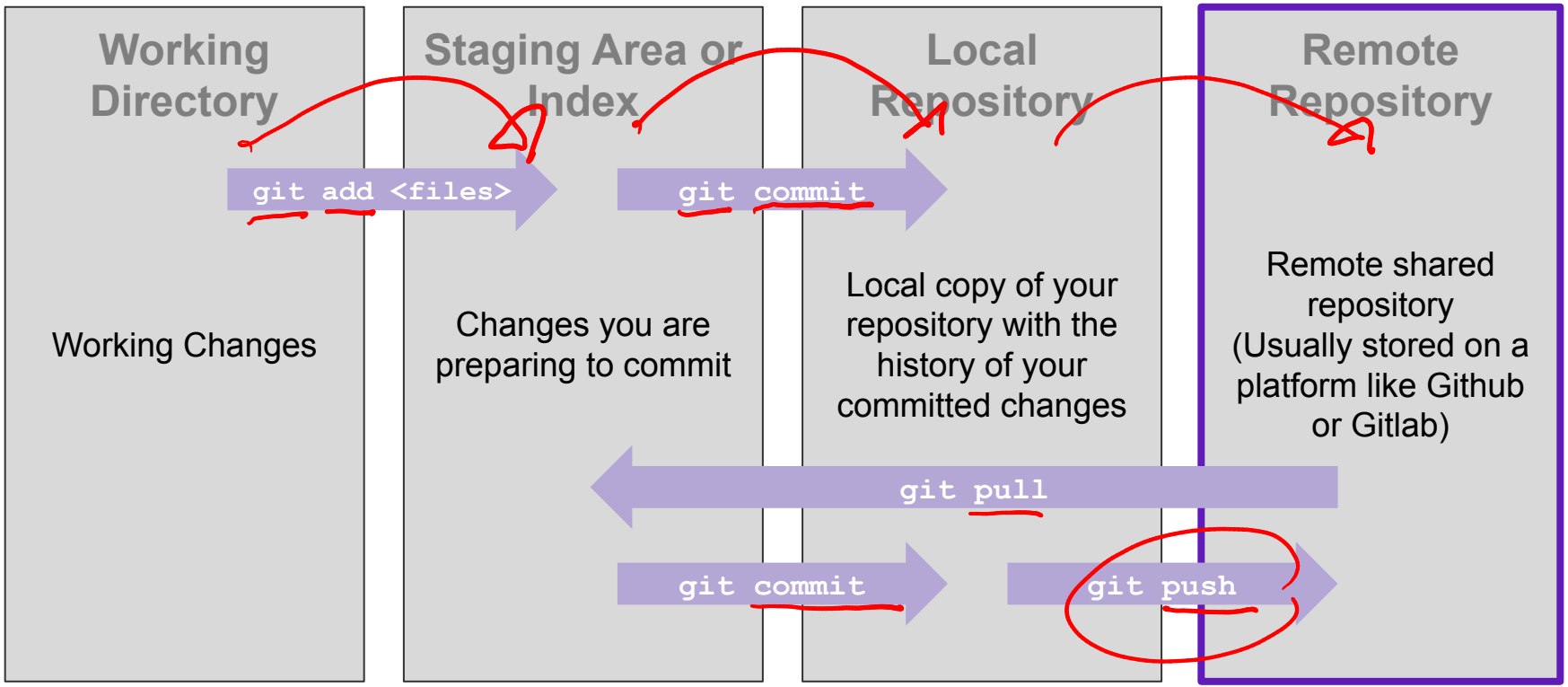
Git: Four Phases



Git: Four Phases



Git: Phase-Changing Commands



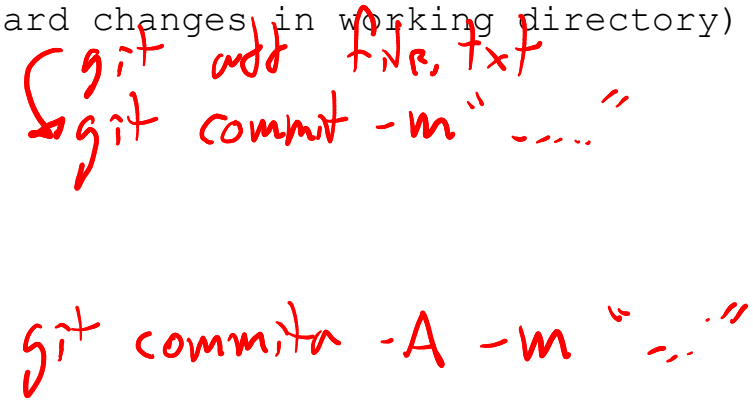
Polling Time!

Please answer in the Poll: Lec 6 assignment on Gradescope!

If `git status` shows the following, which phase is `file.txt` in?

```
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified: file.txt
```

- Working directory
- Staging area / index
- Local repository
- Remote repository



Git Commands

git init, git clone, git add, git commit, git pull, git push, git status, …

The “git" Command

- The `git` command is the primary way of interacting with git
- You must `cd` into the folder where your repo is stored, or any subfolder of the project root
- Used like any other shell command!

```
[cf2024@calgary cse374-25su-cf2024]$ git status
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean
```

Git Commands

Command	Operation
`git clone url [dir]`	Make a local copy of a remote Git repository for your own changes
`git add file`	Adds a file's changes/contents to the staging area
`git commit`	Records a snapshot of the staging area
`git status`	View the status of your files in the working directory and staging area
`git diff`	Shows diff of what is staged and what is modified but unstaged
`git help [command]`	Get help info about a particular command
`git pull`	Fetch from a remote repo and try to merge into the current branch
`git push`	Upload your local commits to the remote repository/server

Creating a Git Repo

Two common scenarios (only do one of these):

1. To create a new local Git repo in your current directory:

```
git init
```

   - This will create a `.git` directory in your current directory.
   - Current directory becomes project root
   - Then you can add and commit changes in that directory into the repo.

1. To clone a remote repo to your current directory:

```
git clone <url> [localDirectoryName]
```

   - This will create the given local directory, containing a working copy of the files from the repo, and a `.git` directory.

What is a "commit"?

- A ***commit*** is a single set of changes made to your repository
- Also records:
  - The name of the author
  - The date and time
  - A commit message: short sentence describing what that commit did
- Identified by an unique alphanumeric ID, aka "SHA"

lines that changed

```
commit 03b4e058d103e3995f685581a5bafbe4d5bc681e
Author: Chloe <cf2024@uw.edu>
Date:   Mon Jun 30 12:03:49 2025 -0700

    adding announcement
```

Commit Messages

- Commit messages are the way you remind yourself and tell others what you did
- Commit messages should be **descriptive**
  - E.g. "Added test for predicting null string"
  - *not* "changed test"
- Commit messages should be **short/medium length**
  - If you want to know *exactly* what code was changed, you can check the full changes (i.e., the diff)

Commit History

- A repository's history is a series of "commits"
- Each commit makes changes to the files in the repo
- *Commit history* serves as a log of the changes everyone made
- `git log` to view the commit history

Commit early and often!

```
commit 03b4e058d103e3995f685581a5bafbe4d5bc681e
Author: Chloe <cf2024@uw.edu>
Date:   Mon Jun 30 12:03:49 2025 -0700

    adding announcement

commit e840c2fe17e3eda6477e2754ea724152fa2c3103
Author: Chloe <cf2024@uw.edu>
Date:   Mon Jun 30 10:14:19 2025 -0700

    changed Weds to Fri for homework schedule

commit 2dc9d0054c88551de3e03b71216fe8c5a155e29b
Author: Chloe <cf2024@uw.edu>
Date:   Mon Jun 30 09:56:03 2025 -0700

    updating things for lecture 4 and scheduling updates

commit 2e47d026695d84d85861776fd88e380c1a0d04dd
Author: Chloe <cf2024@uw.edu>
Date:   Fri Jun 27 09:48:11 2025 -0700

    updating things for lecture 3 and hw1 release
```

The website is stored as a git repo!
Here's some more of the commits

AS A PROJECT DRAGS ON, MY GIT COMMIT MESSAGES GET LESS AND LESS INFORMATIVE.

Source:
XKCD

Add and commit a file

The first time we ask a file to be tracked, and every time *before* we commit a file, we must ***add*** it to the staging area:

```
git add path/to/file
```

- Takes a snapshot of these files, adds them to the staging area so it will be part of the next commit.
- Example: `git add hello.c goodbye.c`
  - Adds / stages all of the files in the current directory: `git add .`

git add -A

To move staged changes into the local repo, we commit:

```
git commit -m "<message>"
```

Viewing changes

To view status of files in working directory and staging area: **`git status`**

- Lists the files you've changed but not yet committed
- Lists files added to the staging area
- Indicates how many commits have made but not yet pushed

no git add

To see what is modified but unstaged: **`git diff`**

To see a list of staged changes: **`git diff --cached`**

To see a log of all changes in your local repo: **`git log`** or **`git log --oneline`** (shorter version). Press **`q`** to exit.

Polling Time!

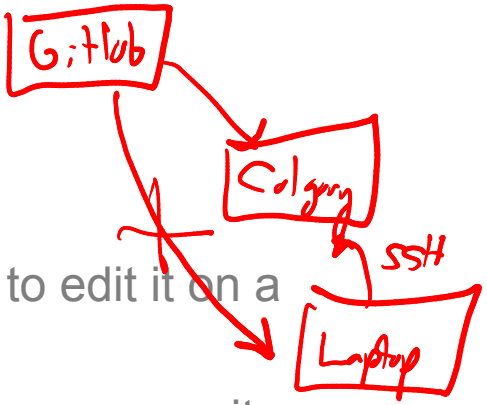
Please answer in the Poll: Lec 6 assignment on Gradescope!

Which of the following is **true** about writing **good** commit messages?

- The first line should be a full paragraph explaining every change in detail
- Messages should describe what was changed, not *why*
- Messages should be concise and summarize intent of the change
- Commit messages are optional and can always be added later

GitLab

CSE Gitlab



- Github and Gitlab are just websites that store git repos
- You can create a repo on the website and `git clone` to edit it on a computer (e.g. your laptop, Calgary, etc.)
- CSE has its own version of Gitlab where you will be given a repository
  - https://gitlab.cs.washington.edu/
  - We'll use this to distribute and submit homework assignments going forward
  - GitLab accounts have been already created for you

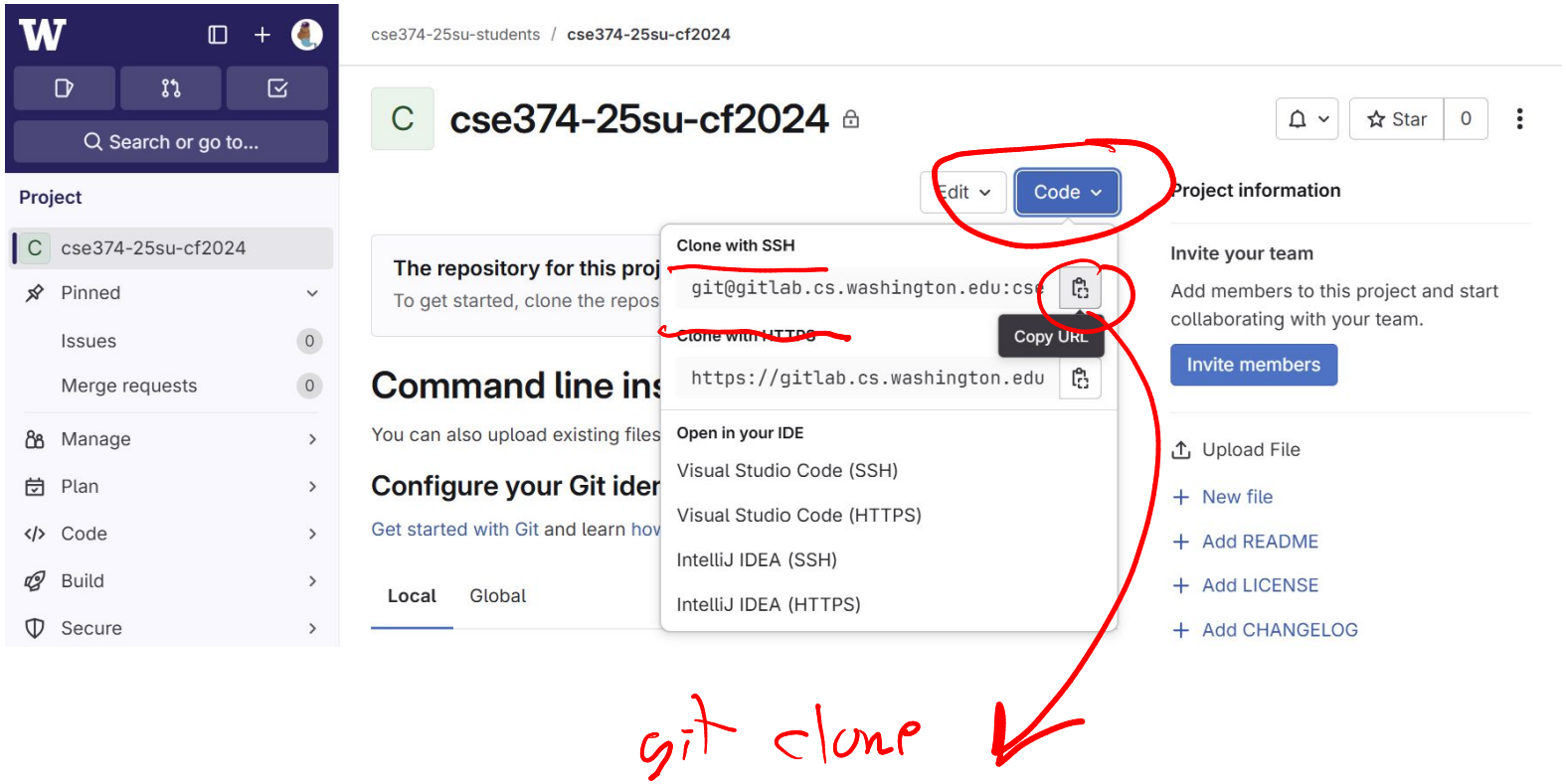
cse374-26sw-UWNETID

GitLab

Getting Started: Instructions also in Ex5

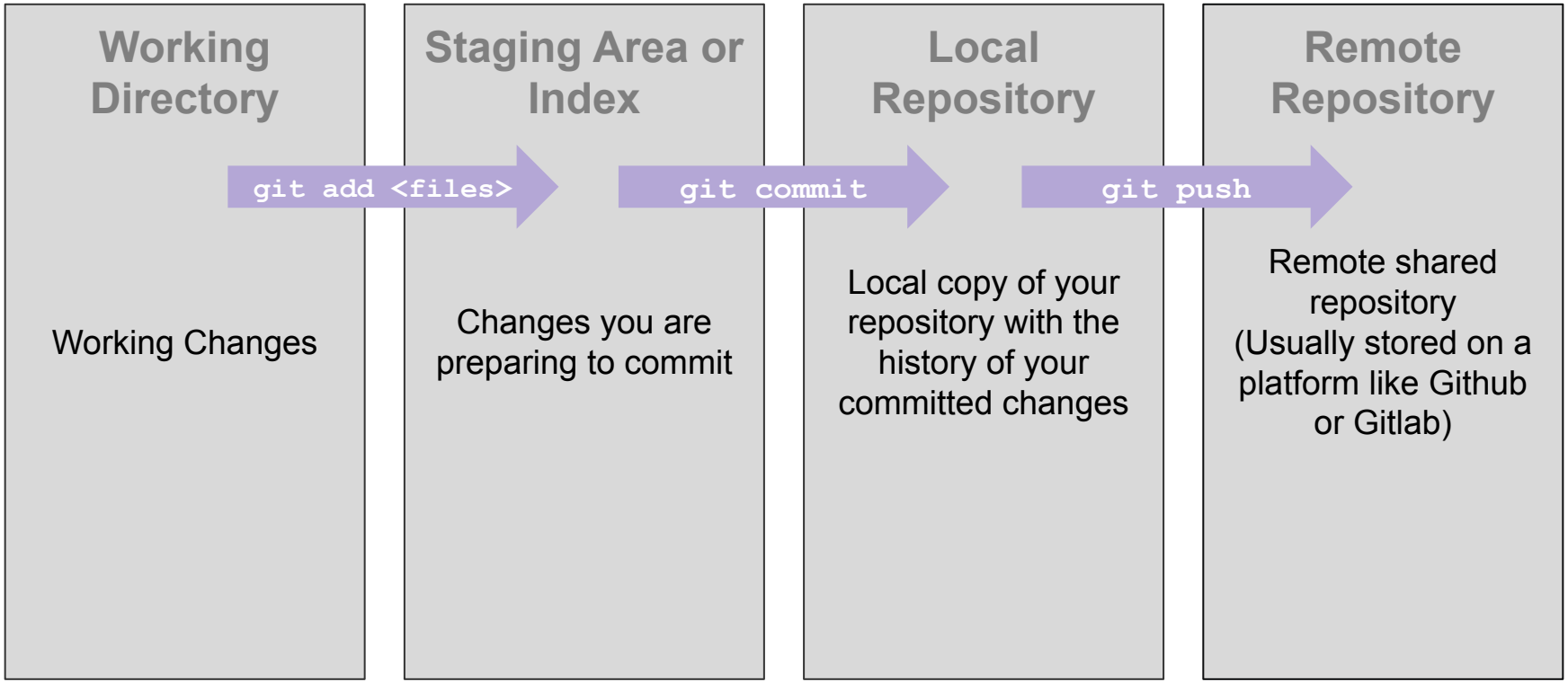
1. SSH into Calgary
2. Create local SSH key (`ssh-keygen`) and add to your GitLab account
3. Clone a local copy of your repository (to Calgary)
   a. `cd <where-you-want-to-put-it>`
   b. `git clone git@gitlab.cs.washington.edu:path/to/repo`
   c. The URL is available from the GitLab page for your homework repo
4. Make sure you use the 'Clone with SSH' URL!
5. If git asks for password, your SSH keys aren't set up right – fix it
   a. Ask us and/or Google if you need help

Cloning From Remote



Demo: `git`

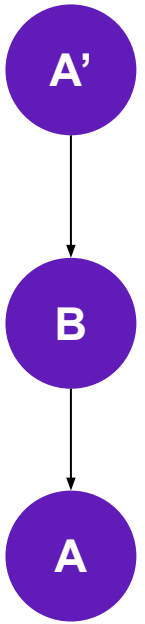
Git: Phase-Changing Commands



Collaborating

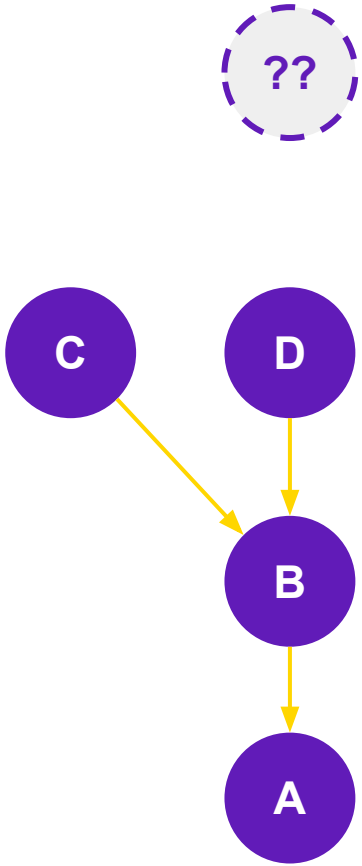
Collaboration: Ideal

- A "linear" history
  - **Alice** makes a commit A and **pushes**
  - **Bob pulls**, makes a change, commits the change in B, and **pushes**
  - **Alice pulls**, makes a change, commits A', and **pushes**
  - ...etc.



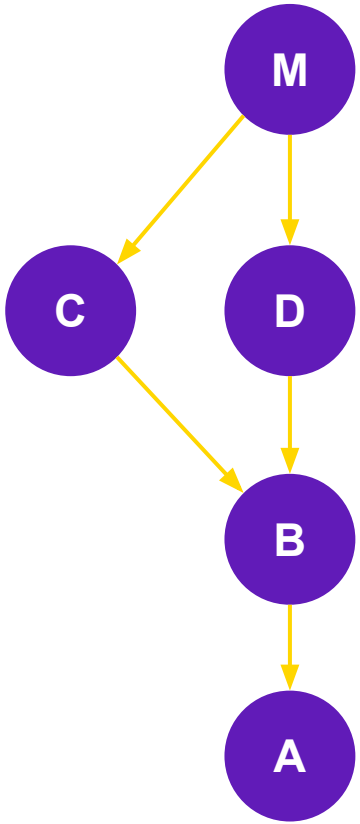
Collaboration: Reality

- We said the "commit history" is a list of commits, so what happens here?
  - **Charlie** makes a change and creates commit C, but **doesn't push**
  - **Diane** also makes a change and commit D, and **pushes**
  - **Charlie pulls** from the remote repo
  - It's no longer a list! The history has diverged
- Does Charlie just have to delete, pull and start over?



Merging

- A merge commit is a commit that has two "parents"
  - Combines the changes in each
  - Commit "M" includes all of Diane's changes, *plus* all of Charlie's



How do we merge?

`git pull`

- Automatically fetches the changes and merges them into yours
- Then, `git push`
  - This push your local changes to the remote repo
  - Others can now work off of your combined changes

Sometimes, the changes you make will ***conflict*** with the changes others make

- e.g. you both edit the same line
- Resolving merge conflicts is more complicated; we will show an overview here but it takes a lot of practice - come to OH or post on Ed!

Merge conflicts

Git will tell you which files had merge conflicts (use **`git status`** to see conflicts), and the files will be edited to identify the conflict:

```
<<<<<<< HEAD:index.html
<div id="footer">todo: message here</div>
=======
<div id="footer">
  thanks for visiting our site
</div>
>>>>>>> SpecialBranch:index.html
```

branch 1's version

branch 2's version

Find all such sections, and edit them to the proper state (whichever of the two versions is newer / better / more correct). You must modify the section to contain the code you want, remove the <<<<<, ======, and >>>>> lines, then save, ***add***, and ***commit*** the merge.

.gitignore

Git may be used to store any types of files.

However…

Do not store files that are unnecessary.

- Backup files (like *.swp vim files, or *~ emacs files)
- Files that can be recreated (such as .o files) should not be added.
- System specific files

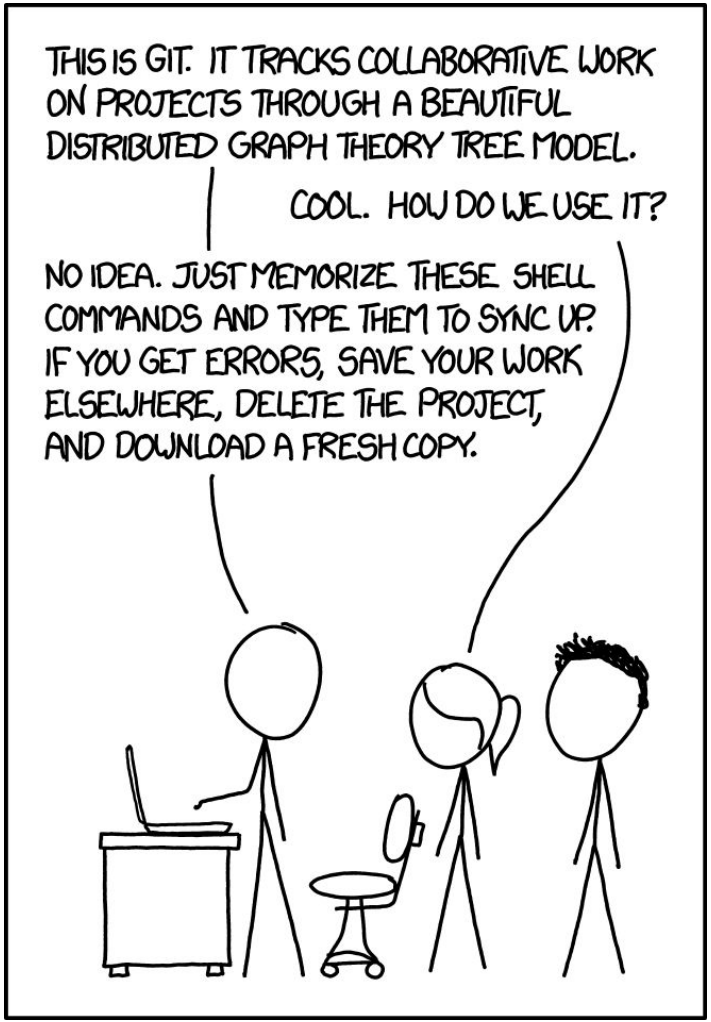
‘.gitignore’ lists files not add to the repo. Below is a sample .gitignore file content:

```
# Ignore vim temporary files
*.swp

# Ignore OS X finder info
files
.DS_Store

# Ignore built object files
*.o
```

How do I fix git?!

You will inevitably run into frustrating situations with git

- Even for experienced users, sometimes you may accidentally get your git repo into an undesirable situation
- There is always a way to fix this, although it's not always obvious how
- Online resources are helpful
    - Stack Overflow, etc.



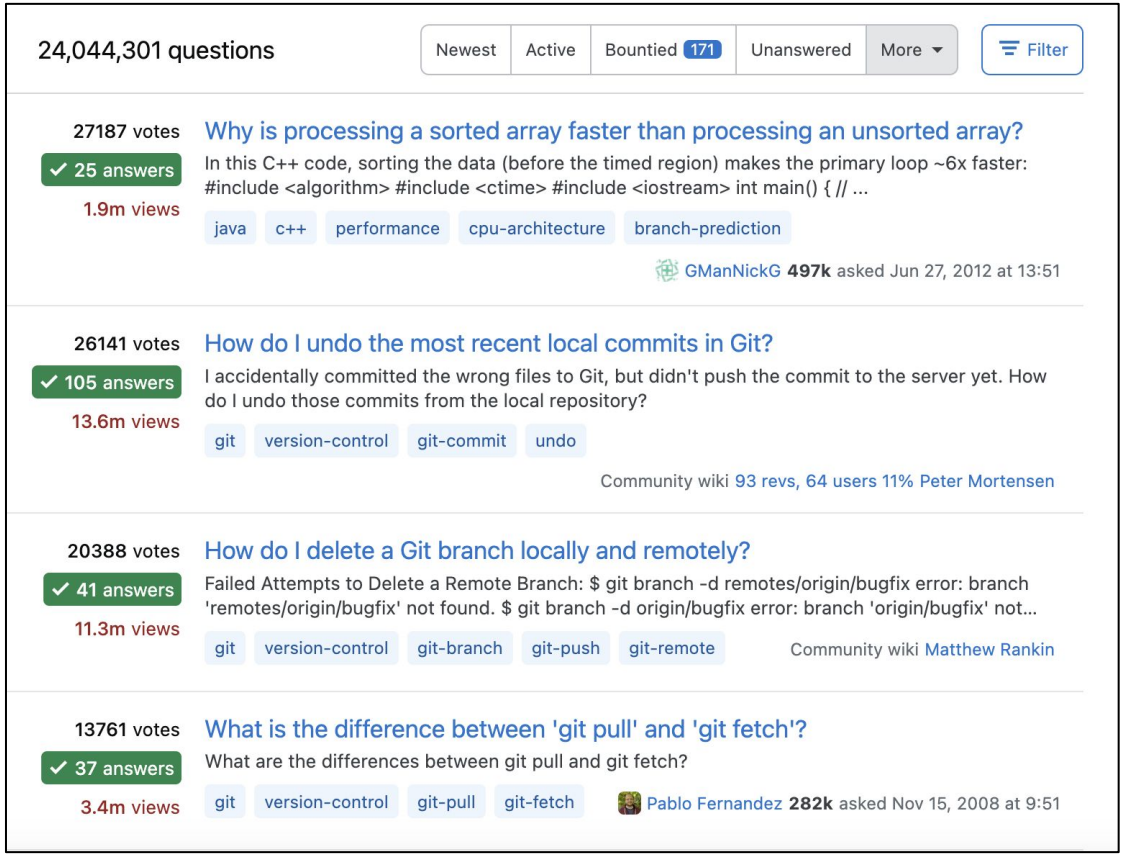
Learning more

- Lots of interesting and useful topics, including:
  - Branching, checkout
  - Resolving merge conflicts, merge tools
  - "Merge requests" (a.k.a. "Pull requests")
  - Rebase
  - `git bisect` - binary search for the commit that introduced a bug!
- The web is your friend!
  - Official documentation
  - "Git Book": https://git-scm.com/book/en/v2
  - Dangit, git!: https://dangitgit.com/en
    - Also available as ohsh*tgit dot com

P.S. git is complex!

Three of the top four most-upvoted questions on StackOverflow.

Everyone is learning!



Assignment Reminders

EX5 due **Friday** 7/11 **@10:49am**

- Ideally should finish by Weds, but giving a bit of extra time in case

HW2: Bash Scripts is **due this Wednesday 7/9 @11:59pm**

- Use what you've learned about Bash commands and scripting to start writing your own short scripts!

Make sure to submit LP6 before the next class as well.

All of these assignments can be found on our course's Gradescope page.

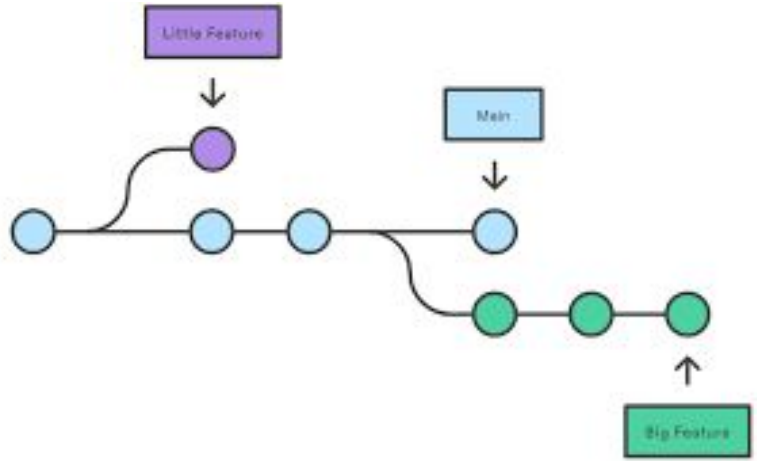
Next lecture: Diving into C programming! 😄

Bonus Slides 

Branch

Git supports branches, which are used to split out a line of work.

- A branch is composed of one or more commits.
- A repository contains one or more branches.



The main branch is the primary source of truth!

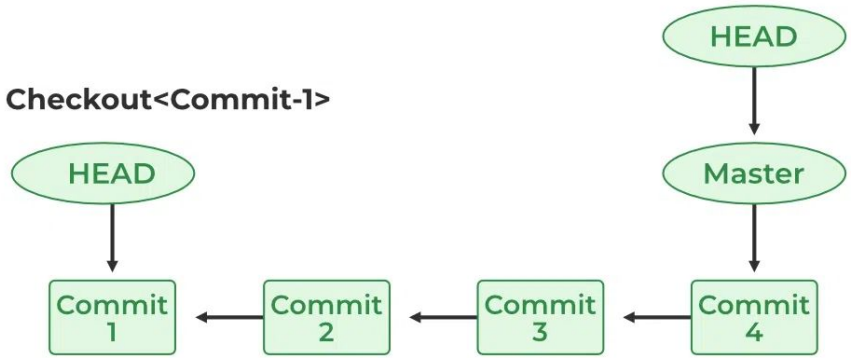
- Default when a repository is created.

repository > branch > commit

HEAD Commit

You will often see references to HEAD, which is the **latest commit**.

- Acts like a bookmark
- Every branch has its own HEAD



Display the content of the latest commit:

```
git show HEAD
```

Link