CSE 391

Introduction to git

Slides created by Josh Ervin. Based off slides made by Marty Stepp, Jessica Miller, Ruth Anderson, Brett Wortzman, and Zorah Fung

AGENDA

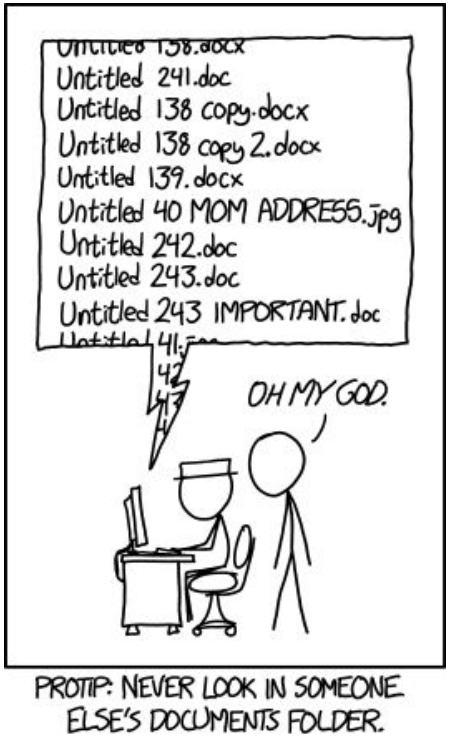
- Logistics, Roadmap
- Introduction to `git`
  - Common git commands
  - Using `git` locally
  - Using `git` with remote
  - Resolving merge conflicts

ROADMAP

- Introduction to the command line
- Input/output redirection, pipes
- More input/output redirection, tee, xargs
- Git: Fundamentals
- Git: Branches and rebasing
- Regular expressions
- More regular expressions, sed
- Users and permissions
- Bash scripting
- Industry applications

VERSION CONTROL - 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 delete a critical file and can't get it back.
  - Your computer was broken or stole 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...



VERSION CONTROL - TEAMS

- Does any of the following sound familiar?
  - My partner and I are paired up for a project for one of our CSE classes. We usually pair program together in the labs 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?
  - How do we keep backups of important files? Who stores them on their computer?

VERSION CONTROL

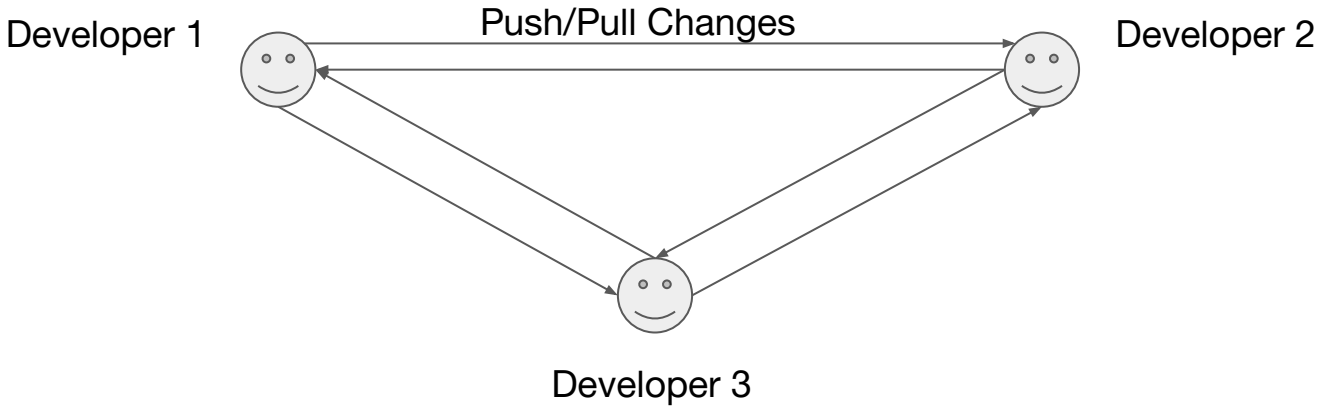
- Version Control: Software that keeps track of changes to a set of files.
- You likely use version control all the time:
  - In Microsoft Word, you might use Ctrl+Z to undo changes and go back to an earlier version of the document.
  - In Google Docs you can see who made what changes to a file.
- Lots of people have a use-case for version control
  - We often think of version control as related to managing code bases, but it's also used by other industries such as law firms when keeping track of document changes over time.

REPOSITORY

- A repository, commonly referred to as a *repo* is a location that stores a copy of all files.
  - The **working directory***(or working tree)* is different from the **repository** (see next slide)
- What should be inside of a repository?
  - Source code files (i.e. .c files, .java files, etc)
  - Build files (Makefiles, build.xml)
  - Images, general resources files
- What should **not** be inside of a repository (generally)
  - Object files (i.e. .class files, .o files)
  - Executables

REPOSITORY

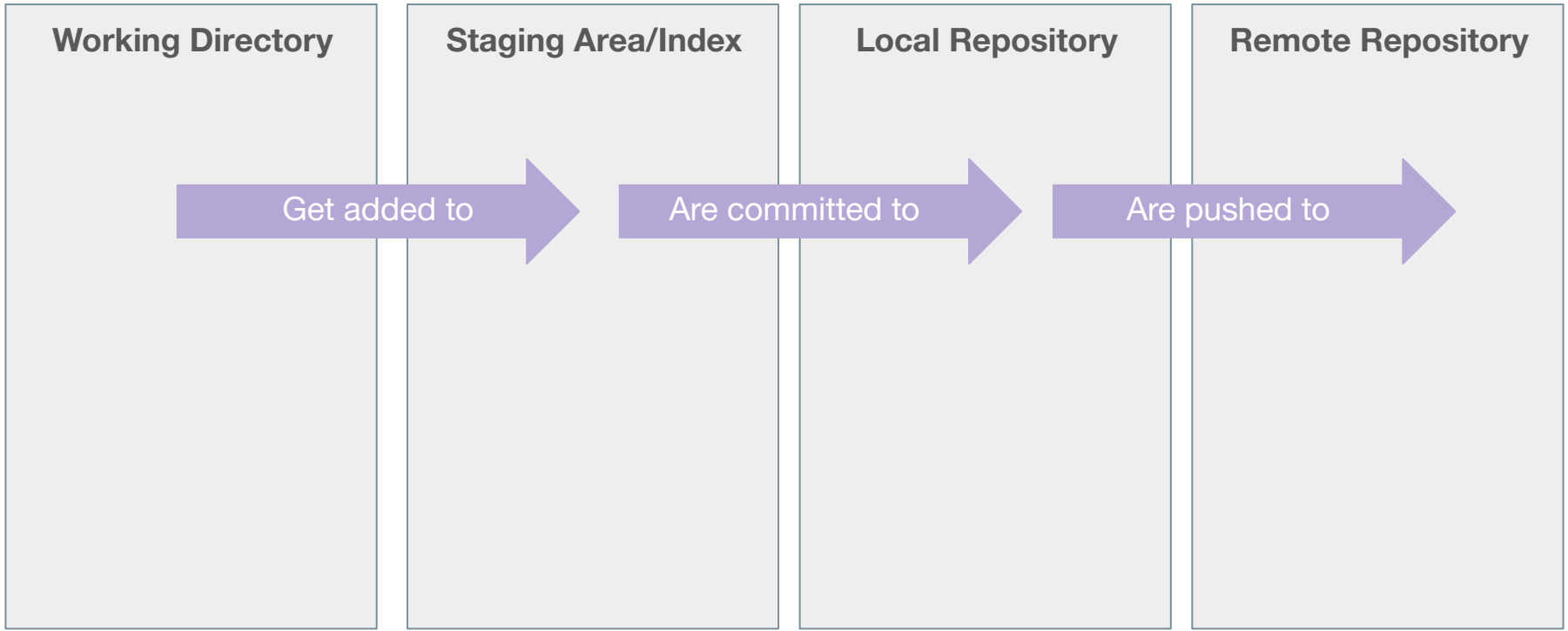
- With `git`, everyone working on the project has a complete version of the repository.
  - There is a **remote** repository, which is the defacto central repository
  - Remote repositories are hosted on services like GitHub or Gitlab
  - Everyone has a local copy of the repository, which is what we use to commit.



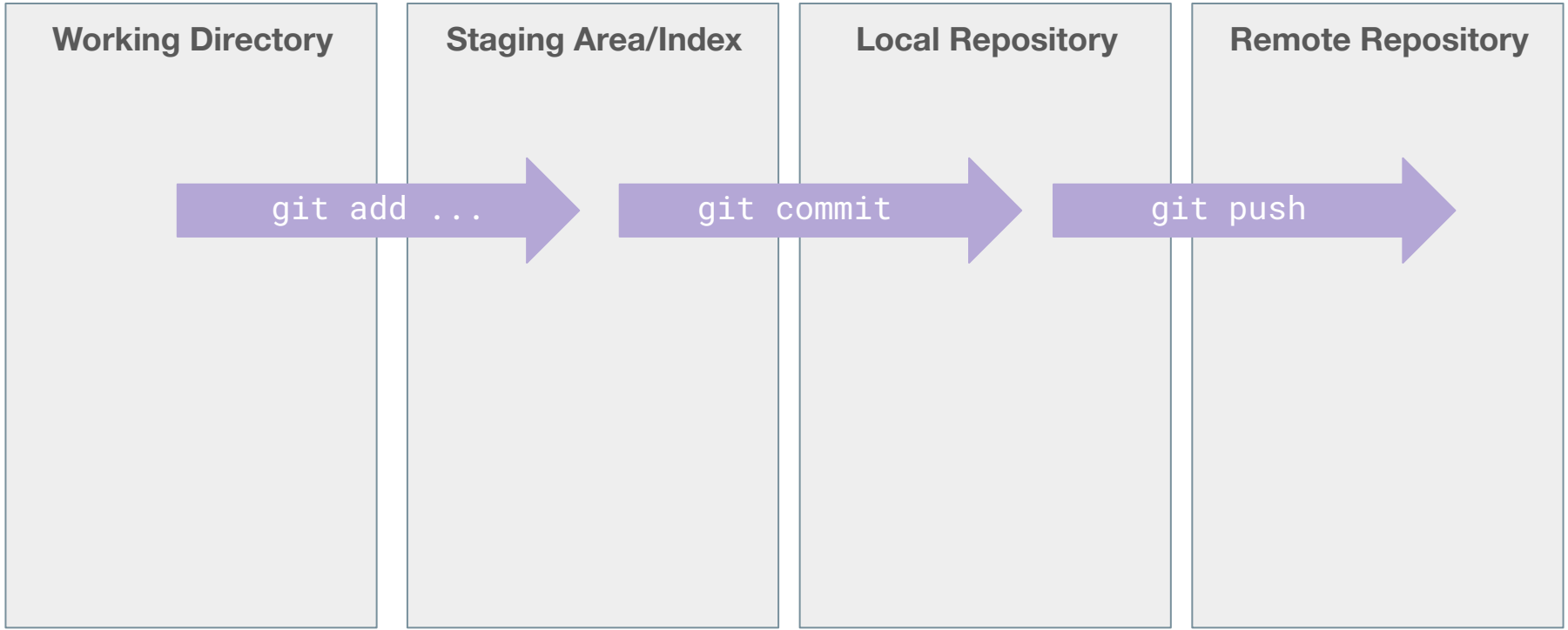
GIT: FOUR PHASES

Working Directory	Staging Area/Index	Local Repository	Remote Repository
Working changes	Changes you're preparing to commit	Local copy of the repository with your committed changes	Remote shared repository (Usually stored with a platform like GitHub/Gitlab)

GIT: FOUR PHASES



GIT: FOUR PHASES



NOTE: There are way more git commands than what is listed here - this is a simplified model to get us started.

INSPECTING A REPOSITORY

`git status`

Working directory

Staging Area

`git log`

Commit History

THINK: pollev.com/cse391

Suppose a run of git status show the following:

```
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:   file1.txt
     modified:   file2.txt
```

Based on this information, at what location are the changes to `file1.txt` and `file2.txt` within the git phases on the previous slides.

A. Working directory
B. Index
C. Local Repository
D. Remote Repository

THINK: pollev.com/cse391

Suppose a run of git status show the following:

```
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:   file1.txt
     modified:   file2.txt
```

We want to run a sequence of commands that includes the changes to `file1.txt` in a single commit followed by the changes to `file2.txt` in another commit in the **local repository**. What commands should we run?

GIT COMMANDS

`git clone` ***`url [dir]`***	Copy a git repository
`git add` ***`files`***	Adds file contents to staging area
`git commit`	Takes a snapshot of staging area and creates a commit
`git status`	View status of files in working directory and staging area
`git diff`	Show difference between staging area and working directory
`git pull`	Fetch from remote repository and try to merge
`git push`	Push local repository to remove repository

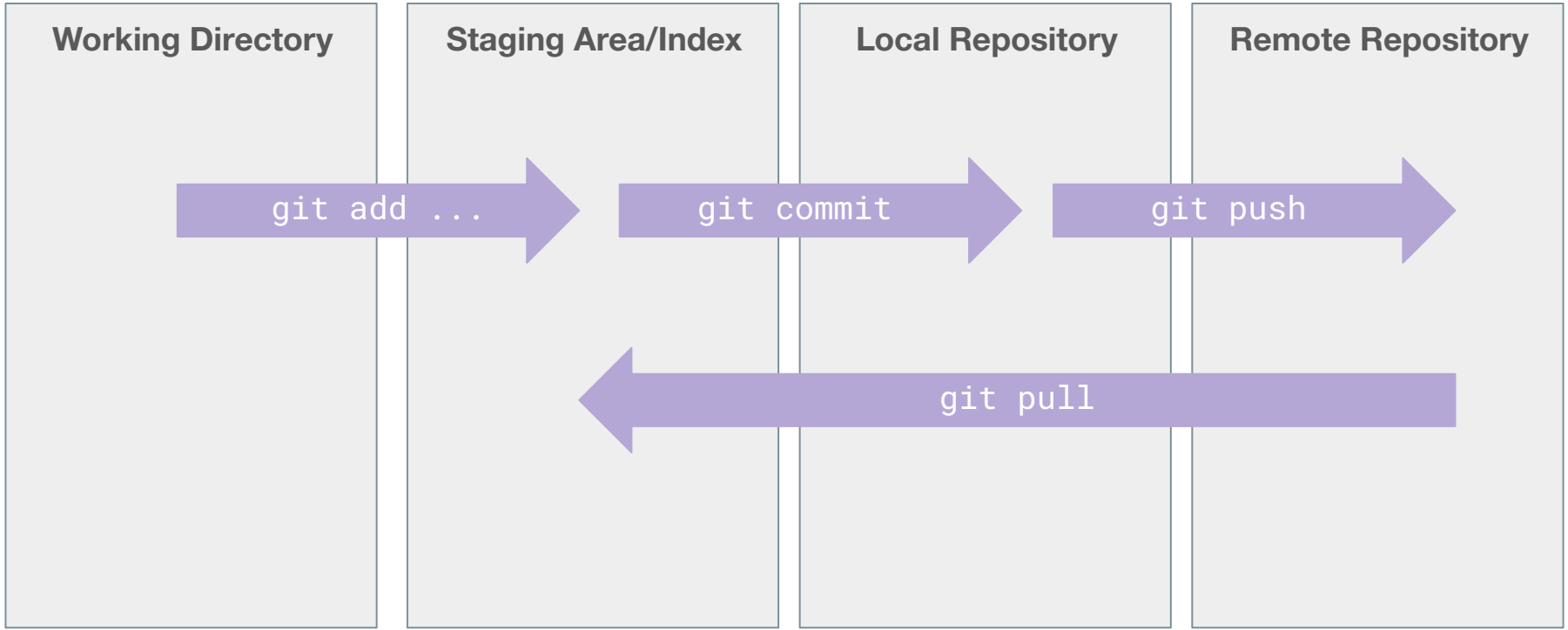
ADDING AND COMMITTING FILES

- The first time we asked a file to be tracked, and everytime before we commit a file we must add it to the staging area. This can be done with the following command
  - `$ git add file1.txt file2.txt`
- This takes a *snapshot* of the files and adds it to the staging area. You can still modify files in the working directory, but you will need to add again to have these changes saved in the staging area.
- *Note: To unstage a change, you can use the following*
  - `$ git reset HEAD filename`
- To move staged changes into the local repository we commit them from the staging area
  - `$ git commit -m “Updated filename”`
- *Note: All of these commands are acting on the local copy of your repository. You will need to push them to remote to see these changes elsewhere.*

COMMIT MESSAGES

- Running `$ git commit` will bring up a text editor by default for you to type your commit message.
  - Write the subject text (less than 50 characters) on the first line, followed by paragraphs describing your commit in the rest of the page.
- Running `$ git commit -m "Your Message"` will allow you to type the message directly in the command line without opening a text editor.
  - This is useful for simple commits, but avoid using it for more complicated commits.
- Regardless, the subject line should always be written with the following form
  - *If applied, this commit* **will your subject line here**

GIT: FOUR PHASES WITH REMOTE



CSE GITLAB

- For this class we will be using gitlab. To access:

1. Log onto CSE Gitlab (https://gitlab.cs.washington.edu)
   a. Use your CSE NetID if you have one
2. Add an ssh key (https://gitlab.cs.washington.edu/help/ssh/README.md)
   a. Follow the instructions in README.md which say to type
      i. `$ ssh-keygen -t rsa -C "yourUWNetID@uw.edu" -b 4096`
      ii. Hit return to accept the default file location. You do not need a password, so you may hit enter twice when prompted for a password
   b. Then type `$ cat ~/.ssh/id_rsa.pub`
   c. Copy and paste the key into the SSH-Keys section under Profile Settings in your user profile on Gitlab.