CSE 374: Programming Concepts and Tools

Spring 2026
Instructor: Megan Hazen

Today's Goals

Subject Matter

- Sed review
- Version Control
- gitlab

Your Goals

- Find your gitlab account
- Clone the CSE 374 repo
- Start your own repo

Aside: GNU

GNU's Not Unix

- Goal is to be completely free
  - Mostly licensed under 'GNU General Public License (GPL)'
- Many software tools
- Typically used with a linux kernel
- Managed by the Free Software Foundation
- https://www.gnu.org/home.en.html

Basic sed (more)

```
$sed [OPTIONS] [COMMAND] [FILE]
$input_stream | sed [COMMAND]
$sed -i 's/orig/replace/g' FILE
```

`FILE:` An input file, could be more than one.

`-i:` an example of an OPTION (flag) – this replaces the original file with an edited one.

`s/orig/replace/g:` the command, enclosed in single quote to avoid globbing. S indicates substitution, orig& replace are regex patterns, g indicates global.

Sed Commands

P : print this line (often used with '-n' to suppress printing of non-marked lines)

d : delete this pattern space and continue

y : transliterate characters

a: append text

i : insert text

c : replace text

```
$ echo hello world | sed
'y/abcdefghij/0123456789/'
74llo worl3$

$ seq 3 | sed '2i hello'
1
hello
2
3

$ seq 10 | sed '2,9c hello'
1
hello
10
```

SED Demo

```
wget https://courses.cs.washington.edu/courses/cse374/26sp/assignments/numberslist
```

Sed additional thoughts

- Sed encounters one line at a time
- and does one pass of the input.
- Delimiter '/' can be changed to anything, like '_' or ':'
  - may help if COMMAND contains many '/'
- Multi-line editing is possible, but painful, with sed (with 'hold buffer'). Use another scripting program (like 'awk').
- Branches are also possibly ('b' and 't' commands)
- Use backreferences (\1, \2 etc) to refer back to regex gathered with \(to \)

What about awk? (or perl? Or ed? Or ruby?)

- `awk` allows you to edit an input stream with more power than `sed`
- All of these, though, are scripting languages with their own flavor and their own power. `sed` is one of the most basic.

What is version control?

- Keeps track of files
- Tracks changes to files
- Manages sharing of files
- Merges multiple and sometimes conflicting updates

Modern version control?

- Keeps track of files
  - Backs up to cloud
- Tracks changes to files
  - Keeps history and traces changes to user
- Manages sharing of files
  - Branching and Merging
  - Rollback and recovery
  - Collaboration and merge conflict resolution

Have you heard of this before?

Perforce

Apache SVN

Unity

Azure

Mercurial

Git

Why is version control?

Backups: Archive a project to keep a safe copy

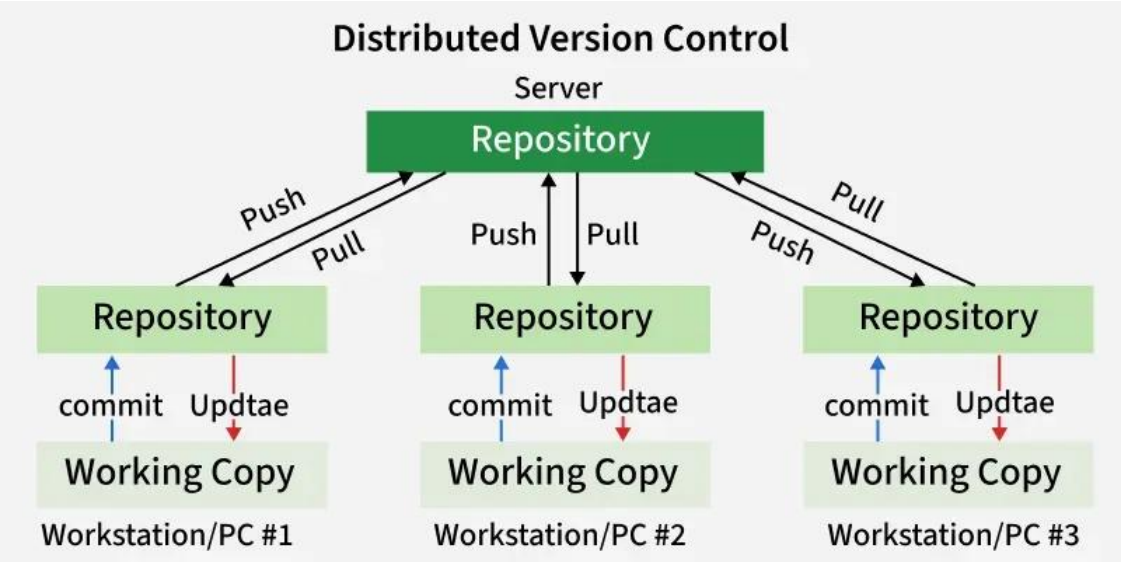
Collaboration: Shared copy, manages concurrent and maybe conflicting changes

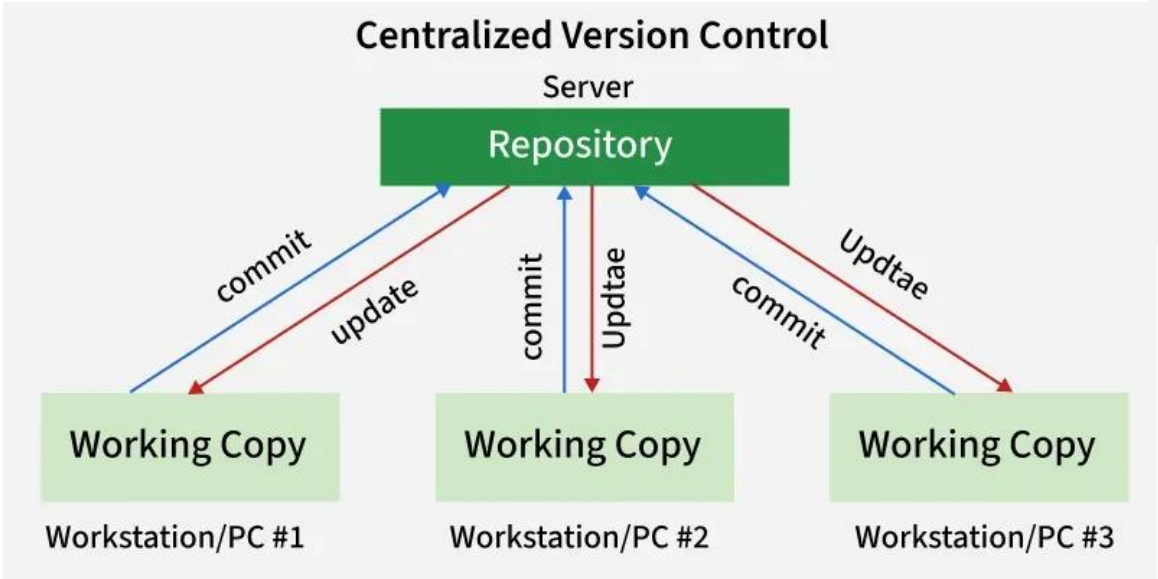
Version log: Keeps copies of previous versions and changes made

Note: This is not software only – version control can be used for all types of documents

Models of version control

- Local
- Centralized (Subversion)
- Distributed (git)

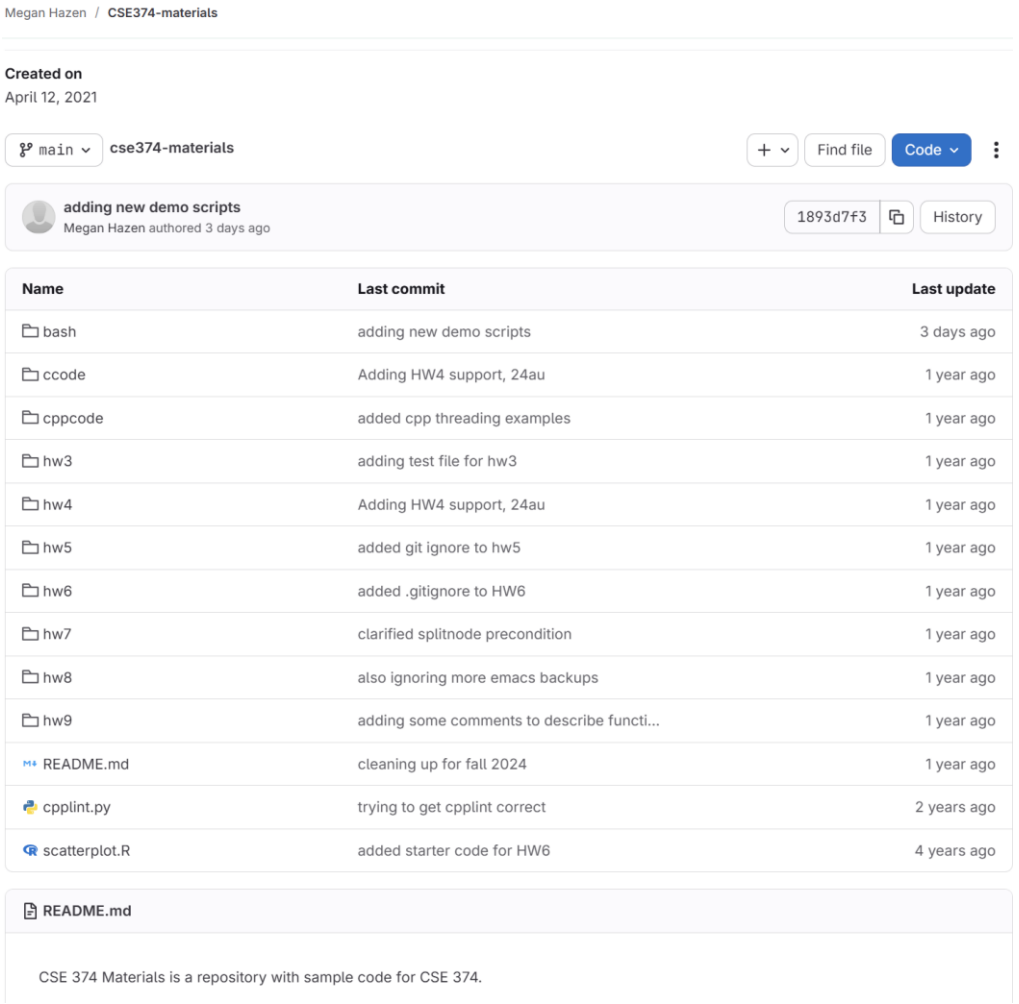




https://www.geeksforgeeks.org/git/version-control-systems/

Repositories (repos)

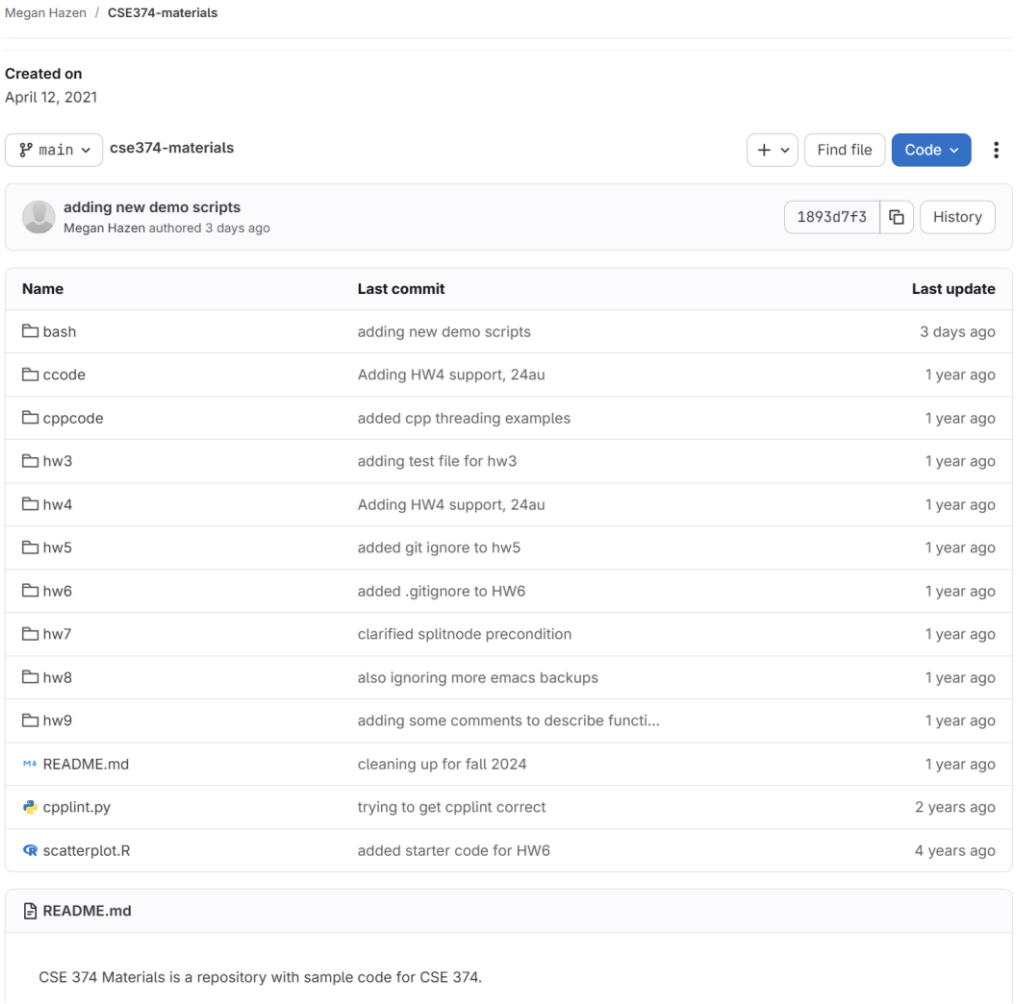
- A repository is a collection of all the files.
  - Consider it a set of nested folders and files
  - Or one branch of the file system tree
- Also includes a history of changes



Repos store

- Source code
- Build files
- Resources files (pictures, data files)

- NOT derived files (object files, backups)

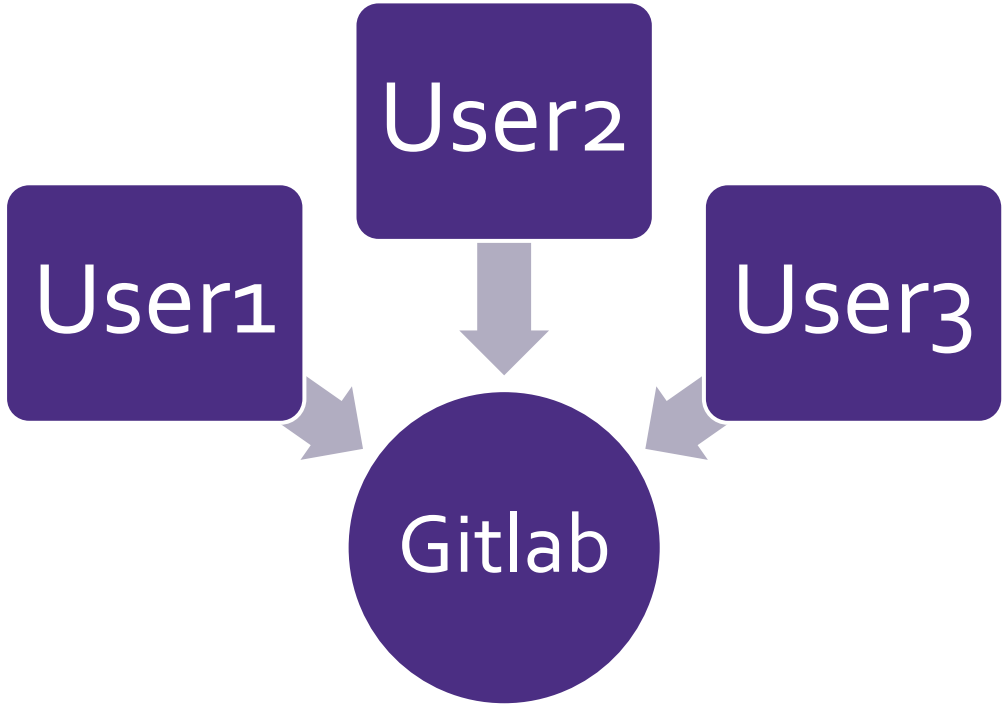


What is git?

- Git is a free and open source distributed version control system
  - We'll use git on Calgary
  - It is already installed.
- Gitlab provides a cloud-based system that hosts gits repositories.
  - Gitlab offers many features beyond basid git, including a web interface.
- Github and Bitbucket are also services that host git repositories

Currently most popular version control software
Developed in 2005 to support Linux work
Both Linux and Git were designed by Linus Torvalds – open source development model

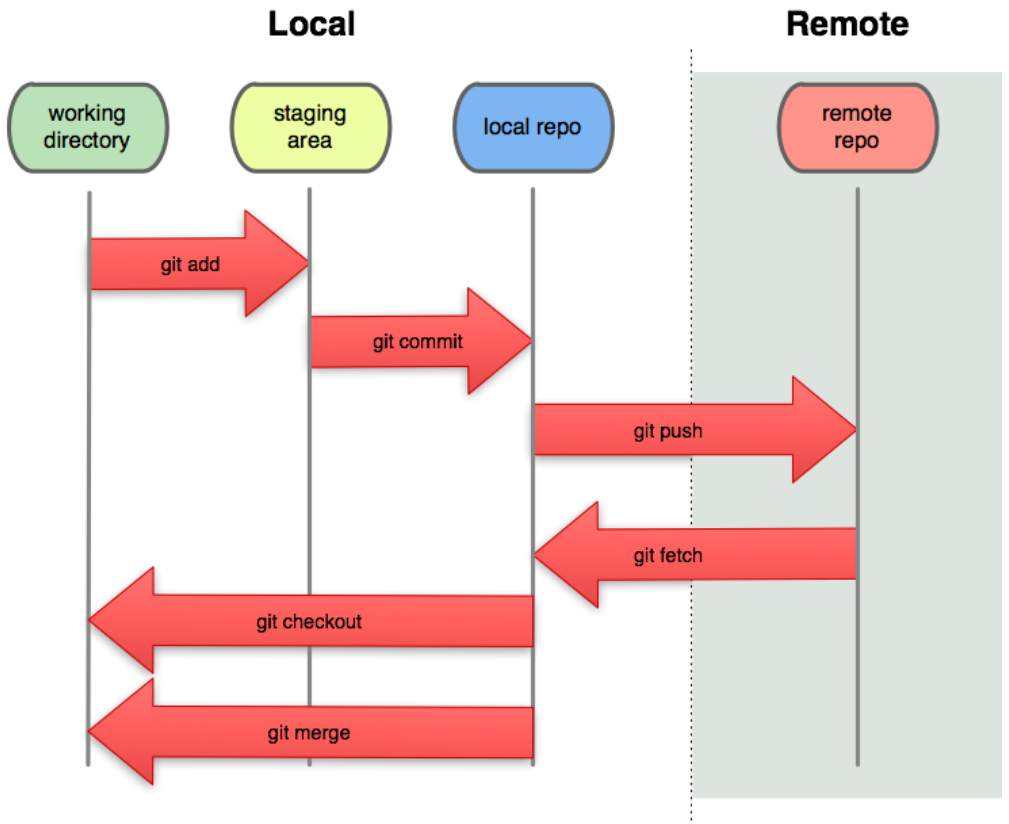
Using your repo



- Gitlab server maintains central copy of each repo (the '`origin`')
- Users '`clone`' (get a copy of) in their local account (on Calgary)
- Users get changes from the origin using `pull`
- Users put their local changes back into origin using `push`

Using your local repo

- On your local machine you can make changes to your local copy of the repo.
- Adding, and committing, changes the status of your local repo.
- In order to put those changes back on the cloud, to back up or share with others, you need to do git push.



Cloud based back up

- Re-access anywhere
- Safe against local hardware damage
- Can share with others

Git command

- The git command lets you interact with git
- **In a folder that is a repo you can use this like any other shell command**

```
[mh75@attu2 cse374-materials]$ ls -a
.   bash   cppcode     .git  hw4  hw6  hw8  README.md
..  ccode  cpplint.py  hw3   hw5  hw7  hw9  scatterplot.R
[mh75@attu2 cse374-materials]$ git status
On branch main
Your branch is ahead of 'origin/master' by 24 commits.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
[mh75@attu2 cse374-materials]$ git pull
Already up to date.
```

Git commands

Command	Operation
`git clone` *`url [dir]`*	Copy a Git repository so you can add to it
`git add` *`file`*	Adds file 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 new repo

Three common scenarios (only do one of these):

1. To create a new local Git repo in your current directory: **`git init`**
   1. This will create a .git directory in your current directory.
   2. Then you can commit files in that directory into the repo.
2. To clone a remote repo to your current directory:
   1. **`git clone url [localDirectoryName]`**
   2. This will create the given local directory, containing a working copy of the files from the repo, and a .git directory.
3. Use the web interface, and then clone

Add and commit a file

- If you change a file, or create a new file, you:

  `git add filename`

- Takes a snapshot of the file and puts it in the staging area
- To put the changes into the local repo you:

  `git commit –m “<message”`

```
$ git status
On branch main
Your branch is ahead of 'origin/master' by 24 commits.
  (use "git push" to publish your local commits)
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:   README.md
no changes added to commit (use "git add" and/or "git commit -a")

$ git add README.md

$ git commit -m 'Updating README for new
quarter'

[main 63996fc] Updating README for new
quarter

 1 file changed, 4 insertions(+), 2
deletions(-)
```

What is a commit?

- One single set of changes to your local repository
- Also records
  - Name of author
  - Date and time
  - A commit message: a brief description of the change
- Identified by an ID, or "SHA"

```
commit 63996fcdae5603da03f47c5ee78cab11eb68f849 (HEAD
→ main, origin/main)
Author: Megan Hazen <mh75@cs.washington.edu>
Date:   Fri Apr 10 15:27:12 2026 -0700

    Updating README for new quarter

commit 1893d7f3470099159b79ae05290f33ad109f1389
Author: Megan Hazen <mh75@cs.washington.edu>
Date:   Mon Apr 6 14:47:11 2026 -0700

    adding new demo scripts
```

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.

XKCD Commit messages

Know your history

- `git status`
  `git status -s`
  - Lists files with changes
  - Tell you un-pushed commits
- `git diff`
  `git diff –cached`
  - Tells you what you changed
- `git log`
  - Tells you commit history

```
[mh75@attu2 cse374-materials]$ git status
On branch main
Your branch is ahead of 'origin/master' by 25 commits.
  (use "git push" to publish your local commits)

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:   README.md

no changes added to commit (use "git add" and/or "git commit -a")
[mh75@attu2 cse374-materials]$ git status -s
 M README.md
```

Moving or removing

- Once files have been committed to gitlab repository:
  - `git mv files`
  - `git rm files`
- git will make changes locally then update the remote GitLab repo when you push
- Can be fixed if you use regular mv/rm commands, but its awkward

Keep your local repo clean

Commit early and often

- Don't lose your work!

Clean up errors as you go

- Don't let them accumulate

Local repo doesn't affect others

- Can even commit incomplete work
- (could always revert)

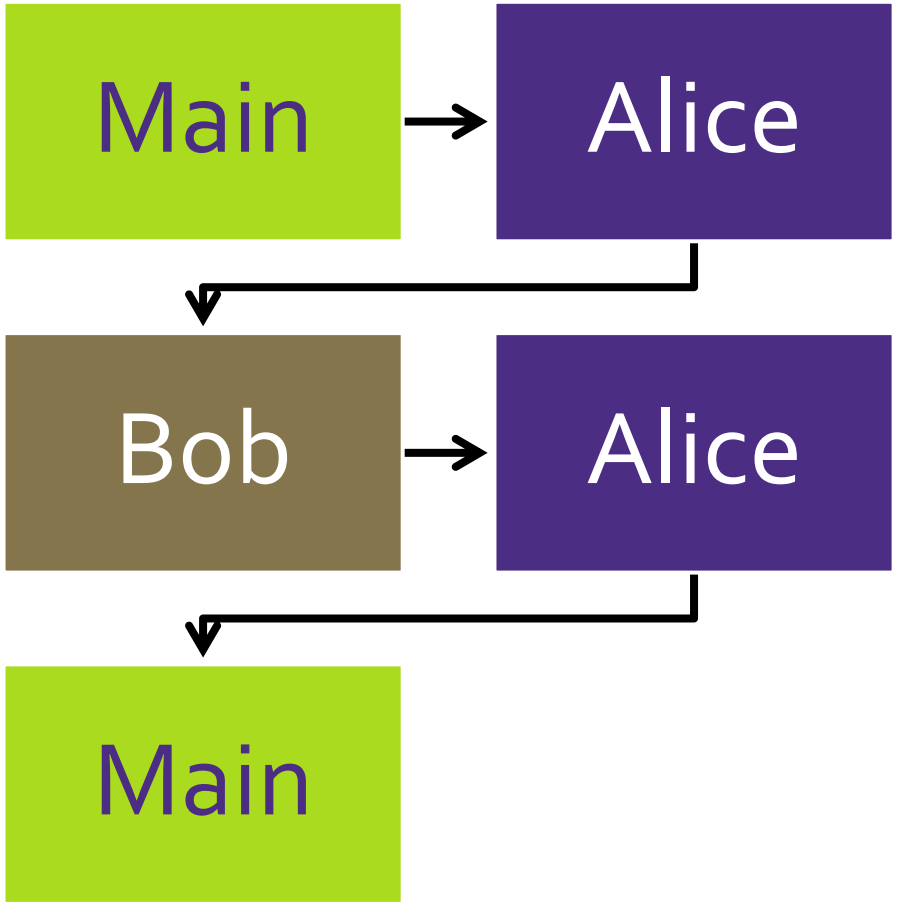
Collaboration!

Use the central server repository to work with others (or back up your latest best copy!)

- Good practice – update with remote changes: `git pull`
  - Also do this any time you want to merge changes pushed by your partner
- Possibly fix any merge messages
- Test, fix anything, `git add / git commit`
- Push accumulated changes to server: `git push`

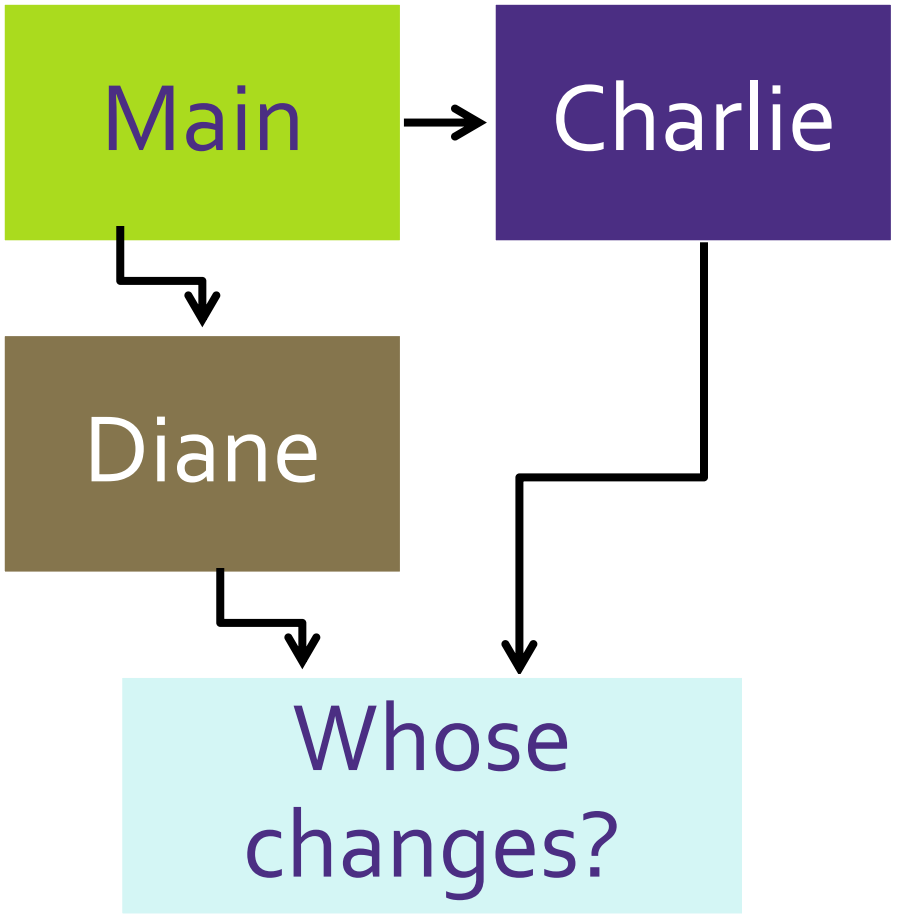
Combining Efforts: Ideal

- **Alice** makes a commit and **pushes**
- **Bob pulls**, makes a change, commits the change, and **pushes**
- **Alice pulls**, makes a change, commits, and **pushes**
- …etc.



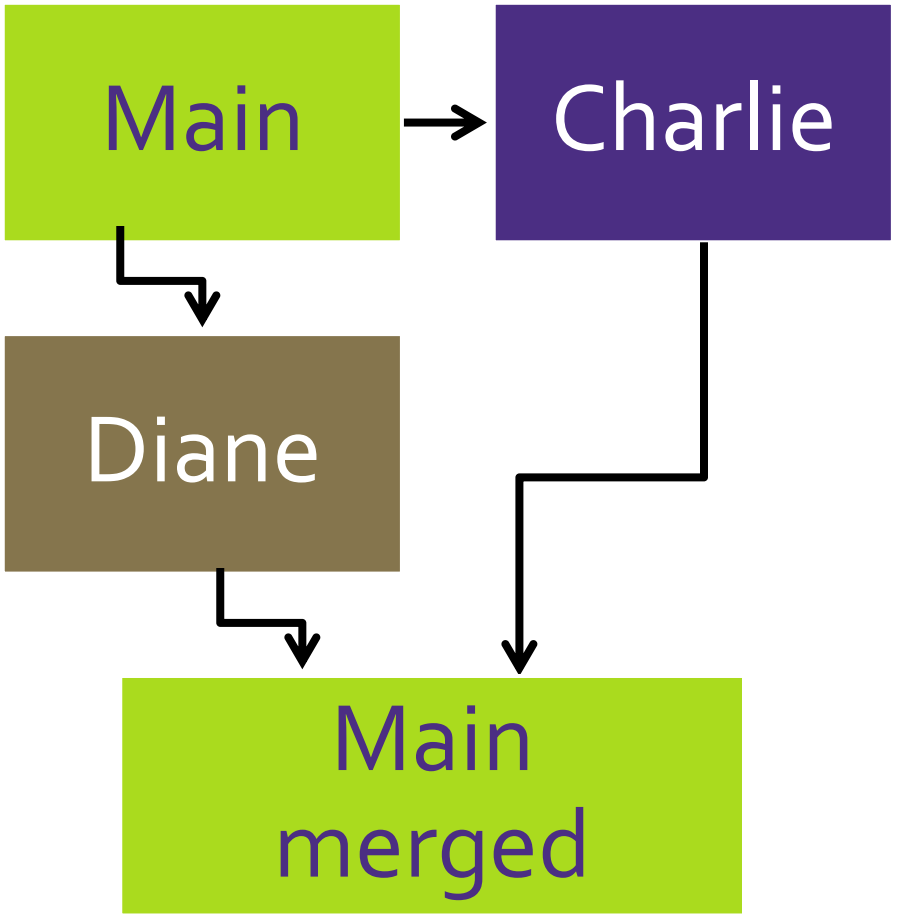
Combining Efforts: Reality

- **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



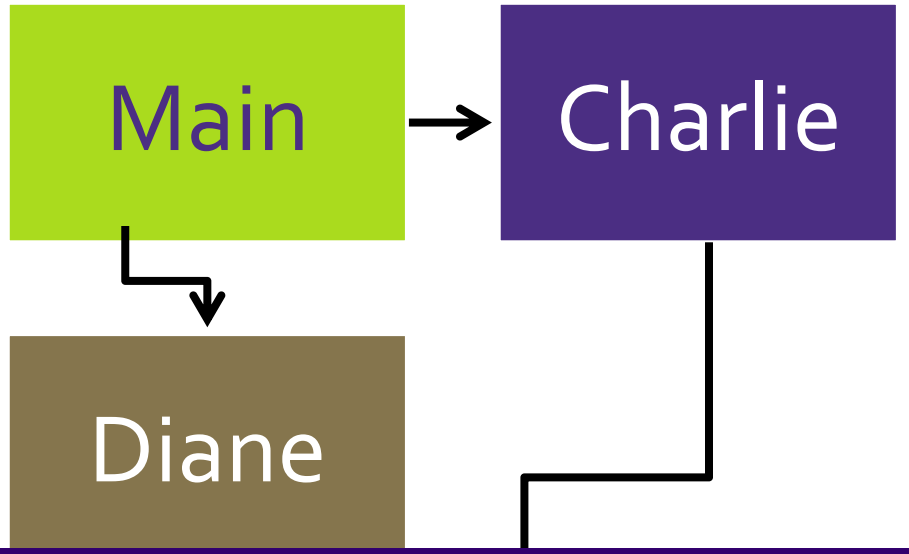
Combining Efforts: Merging

- A **merge commit** has two 'parents'
  - Combines the changes in each
  - Final main contains all of Diane's changes and all of Charlie's changes.



Combining Efforts: Merge Conflict

- A **merge commit** has two 'parents'
  - Sometimes you changed the same line
  - There is a **conflict**
- Edit the file to remove the conflict.
  - Just edit the file removing notes and fixing the issue
  - Recommit and push
  - HEAD is local version




```
<<<<<<< HEAD
The code is gradescope autograder code.  Example solutions may be found in submissions subfol
ders or under homework folder.
=======
The folders primarily contain gradescope autograder code.  Example solutions may be found in
submissions subfolders or under homework folder.
>>>>>>> e570402d9a0953cdc5e9469d6210ab1a76a239d1
```

Branching

'main' refers to the main line, ground truth branch
'HEAD' refers to the branch you are working on in your local repo / the last commit you made
'remote' is the server where the repo is stored

- Git supports branching, which allows you to split out lines of work.
- Branches consist of one or more commits. A repo may have one or more branches.
  - Create a branch to fix a bug `git branch bugfix`
  - Create a different branch to make a new feature `git branch feature`
  - Switch between branches `git checkout bugfix`
- When you are done with the work of a branch, you can re-merge it into Main.
  - Go back to main with `git checkout main`
  - Merge in your bug fix with `git merge bugfix`

Fixing Mistakes

- Can `git checkout FILE` to get the last committed version of the FILE.
- Set local repository to the last commit (forget all changes that you've made), you can run `git reset --hard HEAD`
- Here "HEAD" refers to the most recent commit in your local repository

- Undo a commit using `git revert`
- If the second-to-last commit was bad, you can undo it by saying `git revert HEAD~1`
- Commits aren't completely static and permanent. If you make a commit but then realize you forgot one little thing, you can "amend"/modify your previous commit
  `git commit --amend`

.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)
- Files that can be recreated (such as .o files) should not be added.
- System specific files

'.gitignore' lists files not to upload to HEAD.

```
# Ignore emacs backup files
*.~
# Ignore OS X finder info files
.DS_Store
# Ignore built object files
*.o
```

Fixing Git

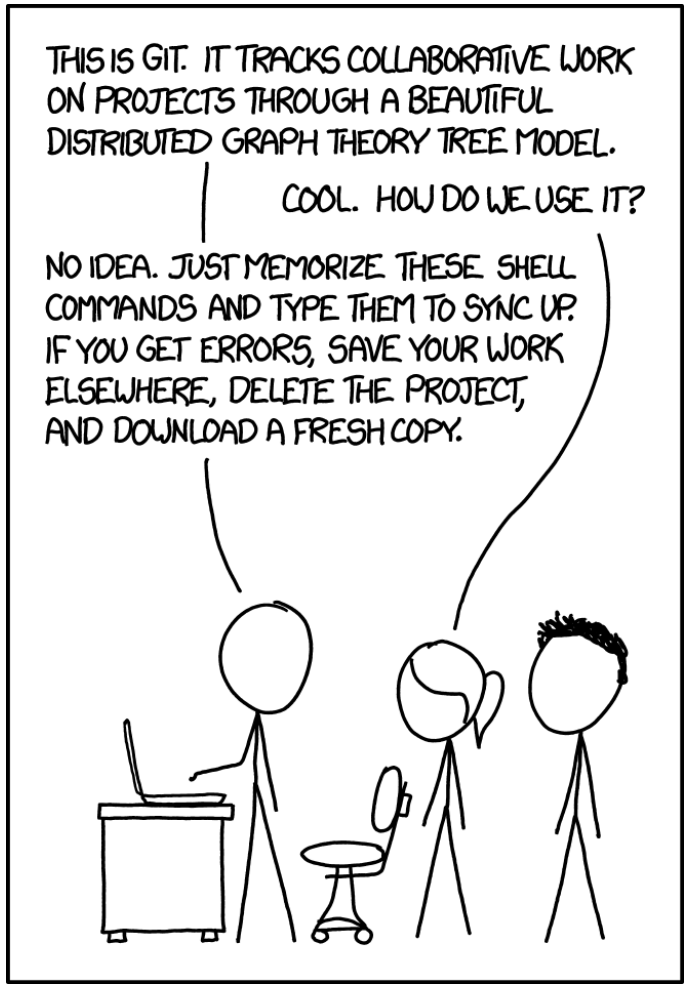
You will inevitably run into frustrating situations with git

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

https://ohshitgit.com/



Week three!

Clone your repo

Push some changes