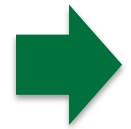


CS 50: Software Design and Implementation

Git and GitHub

Agenda



1. Git vs GitHub
2. Working with a repo
3. Branches
4. Activity

Git is a local Version Control System (VCS)



Person 1

Git is a version control system

- **Runs on your laptop**
- **Keeps local copy of code**
- **Can edit while offline**
- **Change history tracked in local database (.git directory)**

Multiple people working on the same project can run Git at the same time



Person 1



Person 2

...



Person n

Git is a version control system

- Runs on your laptop
- Keeps local copy of code
- Can edit while offline
- Change history tracked in local database (.git directory)

Multiple people can run Git

- Each person keeps a local copy of code on their computer
- Can edit code locally
- Each has history of all changes

GitHub is a cloud-based data repository



GitHub is an online repository

- Stores code
- Can sync to local computers
- There are others (bitbucket)



Person 1



Person 2

...



Person n

Git is a version control system

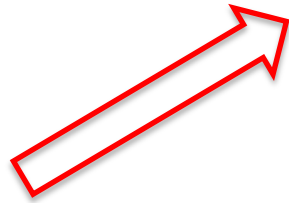
- Runs on your laptop
- Keeps local copy of code
- Can edit while offline
- Change history tracked in local database (.git directory)

Multiple people can run Git

- Each person keeps a local copy of code on their computer
- Can edit code locally
- Each has history of all changes

Changes made locally are pushed to GitHub

“Push” changes made locally to remote repository



GitHub is an online repository

- Stores code
- Can sync to local computers
- There are others (bitbucket)



Person 1



Person 2

...



Person n

Git is a version control system

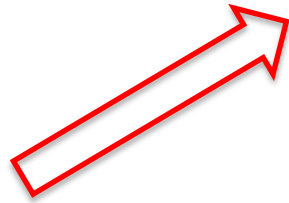
- Runs on your laptop
- Keeps local copy of code
- Can edit while offline
- Change history tracked in local database (.git directory)

Multiple people can run Git

- Each person keeps a local copy of code on their computer
- Can edit code locally
- Each has history of all changes

Other team members can pull the changes from GitHub to stay in sync

“Push” changes made locally to remote repository



- GitHub is an online repository**
- Stores code
 - Can sync to local computers
 - There are others (bitbucket)

“Pull” changes from remote repository to stay in sync



Person 1



Person 2

...



Person n

Git is a version control system

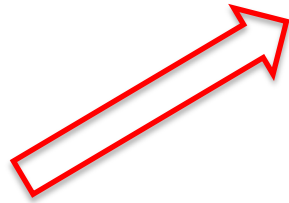
- Runs on your laptop
- Keeps local copy of code
- Can edit while offline
- Change history tracked in local database (.git directory)

Multiple people can run Git

- Each person keeps a local copy of code on their computer
- Can edit code locally
- Each has history of all changes

Other team members can pull the changes from GitHub to stay in sync

“Push” changes made locally to remote repository



Person 1

Git is a version control system

- Runs on your laptop
- Keeps local copy of code
- Can edit while offline
- Change history tracked in local database (.git directory)



Person 2

Multiple people can run Git

- Each person keeps a local copy of code on their computer
- Can edit code locally
- Each has history of all changes

GitHub is an online repository

- Stores code
- Can sync to local computers
- There are others (bitbucket)



Person n

“Pull” changes from remote repository to stay in sync

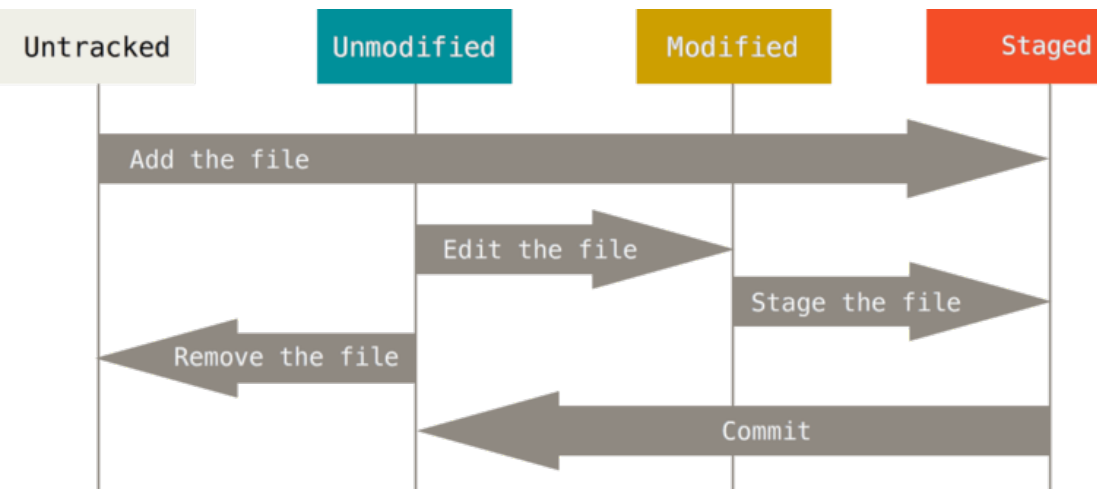
All local databases reflect changes, even if changes made by someone else

Files in Git can be one of several states; initially they are untracked

Git on your laptop



Files are not tracked by Git unless you tell it to track them



Add/Delete/Modify files in local working directory

Use `git add` to tell Git to track new files or to stage changes to existing files

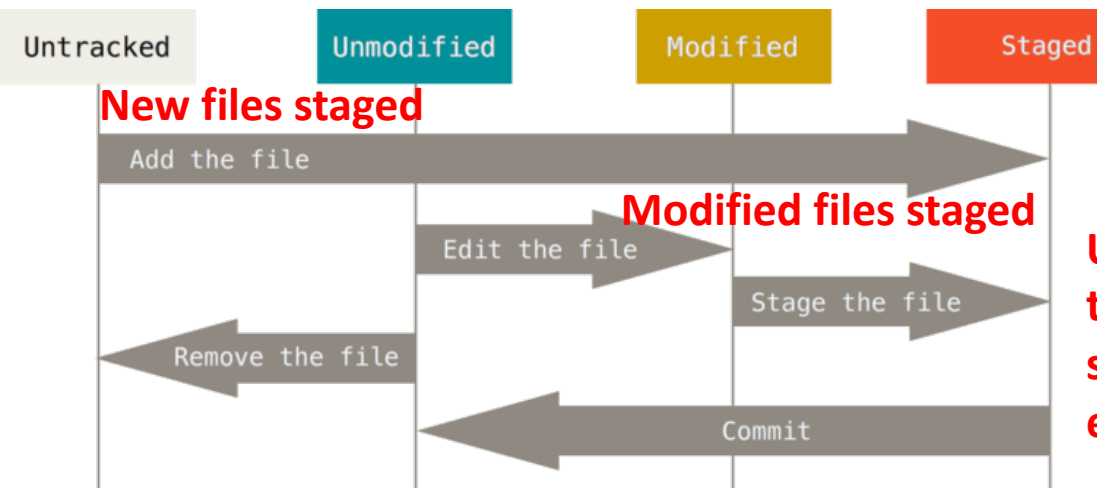
Git on your laptop



Example:

`git add README.md`

README is now staged to be committed



Use `git add` to track new files or stage modified existing files

Add/Delete/Modify files in local working directory  **Stage updates to go into next commit**

Use git commit to take a snapshot of the code in your local database

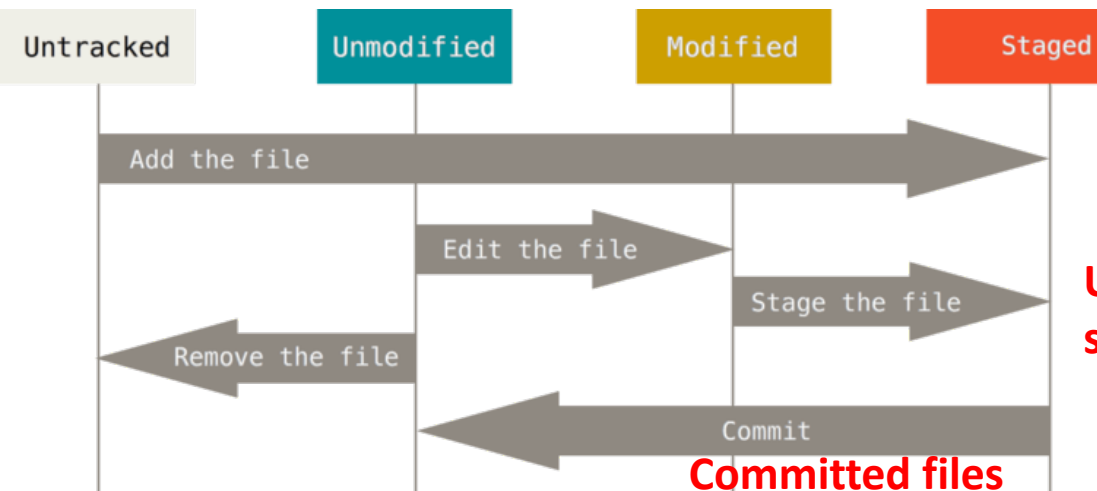
Git on your laptop



Example:

`git commit -m "message"`

Git takes a snapshot of the added files, stores in local database



Use git commit to save snapshot

Add/Delete/Modify files in local working directory



Stage updates to go into next commit



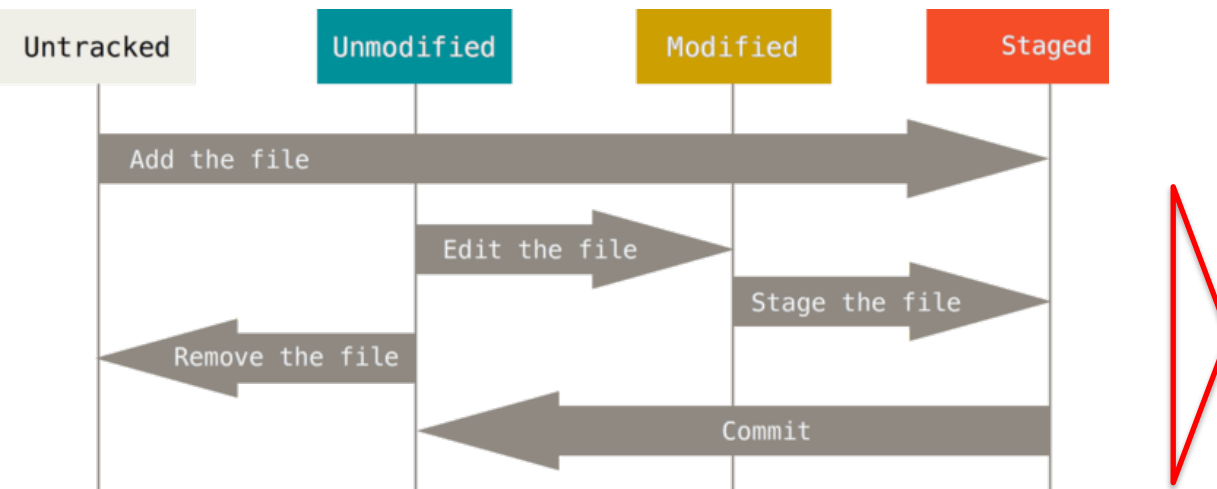
Commit staged changes to local git database (.git/)

Use git commit to take a snapshot of the code in your local database

Git on your laptop



Example:
`git push -u origin main`
Send changes to remote



After completing milestones, push code to Github repository in the cloud

Add/Delete/Modify files in local working directory

Stage updates to go into next commit

Commit staged changes to local git database (.git/)

Use git commit to take a snapshot of the code in your local database

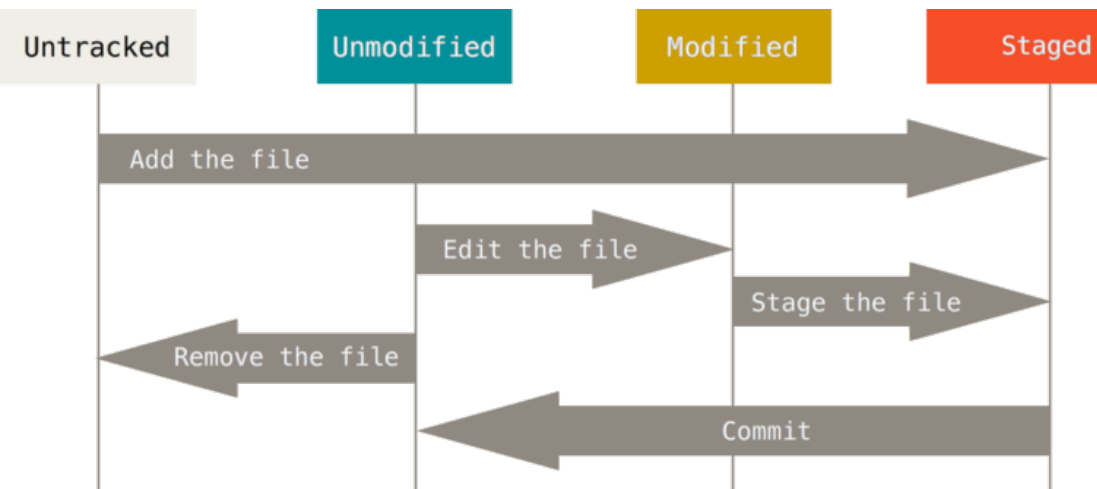
Git on your laptop



Example:

`git pull`

Update local code with remote repo



After completing milestones, push code to Github repository in the cloud

Other team members can now pull your changes and stay in sync

Add/Delete/Modify files in local working directory

Stage updates to go into next commit

Commit staged changes to local git database (.git/)

DO NOT edit files directly in GitHub web interface, use push and pull!

Agenda

1. Git vs GitHub

 2. Working with a repo

3. Branches

4. Activity

Starting a new project with Git begins with `git init` in the working directory

Make a directory for a project and change into it

```
$ cd ~/cs50/activities/day13
```

```
$ mkdir demo1
```

```
$ cd demo1
```

```
$ git init
```

```
Initialized empty Git repository in  
/thayerfs/home/d84xxxx/cs50/activities/day13/demo1/.git/
```

```
$ ls -a
```

```
./ ../ .git/
```

Initialize Git in directory
Git creates subdirectory called `.git`

`.git` directory holds git preferences for this project and database of changes to files

Currently no files are tracked by Git

Git only tracks files we tell it to track


Use `git clone <repo URL>` to begin with existing code already on GitHub

Files added to the working directory are not tracked unless we tell Git to track them

```
$ cd ~/cs50/activities/day13
$ mkdir demo1
$ cd demo1
$ git init
Initialized empty Git repository in
/thayerfs/home/d84xxxx/cs50/activities/day13/demo1/.git/
$ ls -a
./ ../ .git/
$ vi README.md
```

Add a README.md file

Not tracked unless we Git to track it



git status gives the current state of the working directory

```
$ git status
```

```
On branch main
```

```
No commits yet
```

```
Untracked files:
```

```
(use "git add <file>..." to include in what will be  
committed)
```

```
README.md
```

 Git knows README.md exists, but
is not tracking it

```
nothing added to commit but untracked files present (use  
"git add" to track)
```

Use `git add` to track files to tell Git which files to track

Tell Git to track README.md

To track all files, use `git add .`

```
$ git add README.md
```

```
$ git status
```

```
On branch main
```

(Well not quite all files, `.gitignore` is discussed shortly)

Check status again

```
No commits yet
```

README.md is now *staged* to be saved in next commit (snapshot)

```
Changes to be committed:
```

```
(use "git rm --cached <file>..." to unstage)
```

```
new file:   README.md
```

Not saved to history yet!

Use `git commit` to save a snapshot of the code in the local history

Commit writes change to local database

Can think of this like taking a snapshot

`-m` flag provides short message about changes made

```
$ git commit -m "Initial commit"
[main (root-commit) 2aec1ce] Initial commit
1 file changed, 2 insertions(+)
 create mode 100644 README.md
$ git status
On branch main
nothing to commit, working tree clean
```

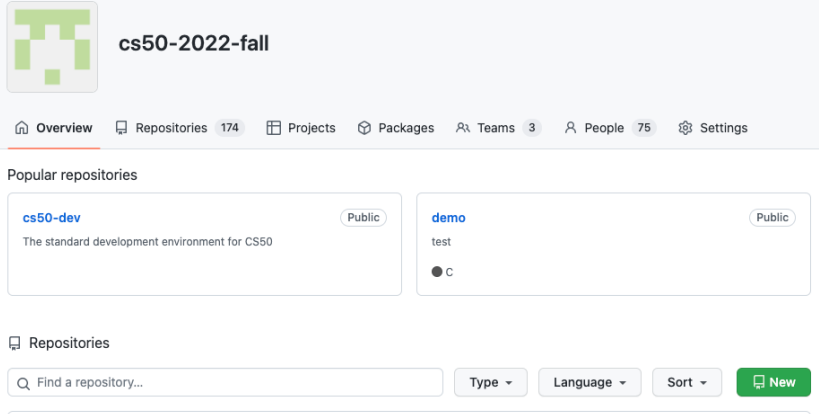
Version are identified
by a SHA-1 hash of files

After commit, snapshot is taken

Git is up to date

Files are still only on local machine, but not
yet pushed to online repo (Github)

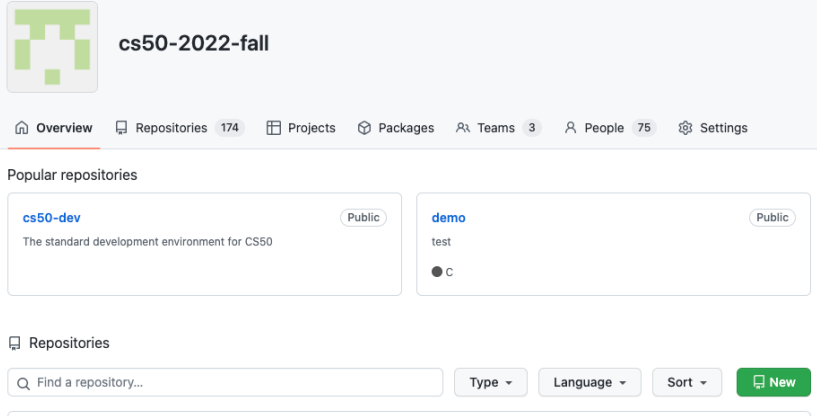
Create a repo on GitHub via web interface



Log into GitHub

Click New to create a new repo

Create a repo on GitHub via web interface



The screenshot shows the GitHub profile page for the user 'cs50-2022-fall'. The profile picture is a green and white pixelated logo. The navigation bar includes 'Overview', 'Repositories 174', 'Projects', 'Packages', 'Teams 3', 'People 75', and 'Settings'. Under 'Popular repositories', two public repositories are shown: 'cs50-dev' (The standard development environment for CS50) and 'demo' (test). Below this is a 'Repositories' section with a search bar and filters for 'Type', 'Language', and 'Sort', along with a 'New' button.

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository](#).

Repository template

Start your repository with a template repository's contents.

No template ▾

Owner *

Repository name *

 cs50-2022-fall ▾

/

Give repo a name

Create a repo on GitHub via web interface

The screenshot shows the GitHub profile for 'cs50-2022-fall'. The navigation bar includes Overview, Repositories (174), Projects, Packages, Teams (3), People (75), and Settings. Under 'Popular repositories', two are listed: 'cs50-dev' (Public) and 'demo' (Public). The 'demo' repository is selected, showing its name and a 'Public' badge. Below this is a search bar for repositories and filters for Type, Language, and Sort, along with a 'New' button.

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Repository template

Start your repository with a template repository's contents.

No template ▾

Owner *

Repository name *

cs50-2022-fall ▾

/

Make sure SSH is selected

The screenshot shows the 'Quick setup' section of the repository creation process. It features three buttons: 'Set up in Desktop', 'HTTPS', and 'SSH'. The 'SSH' button is highlighted with a red arrow. Below the buttons is the repository URL: 'git@github.com:cs50-2022-fall/demo2.git', with a copy icon to its right. A red arrow points from the text 'Copy repo URL' to this copy icon. Below the URL, there are two sections for command-line setup: one for creating a new repository and one for pushing an existing repository. Both sections include a copy icon for their respective code blocks.

Quick setup — if you've done this kind of thing before

Set up in Desktop or HTTPS SSH `git@github.com:cs50-2022-fall/demo2.git`

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

...or create a new repository on the command line

```
echo "# demo2" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin git@github.com:cs50-2022-fall/demo2.git
git push -u origin main
```

...or push an existing repository from the command line

```
git remote add origin git@github.com:cs50-2022-fall/demo2.git
git branch -M main
git push -u origin main
```

Add the repo and push your code to that repo

Currently not configured with a remote

```
$ git remote -v
$ git remote add origin git@github.com:cs50-2022-fall/demo1.git
$ git push -u origin main
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 253 bytes | 19.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To github.com:cs50-2022-fall/demo1.git
 * [new branch]      main -> main
Branch 'main' set up to track remote branch 'main' from
'origin'.
```

Add remote called origin with repo URL

Push code to GitHub repo

Summary

Init

Edit file -> Add file -> Commit -> Push -> Repeat

Others can clone the code from GitHub

On MacBook, not plank

Clone repo using repo URL

```
[MacBook]$ git clone git@github.com:cs50-2022-fall/demo1.git
Cloning into 'demo1'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.
[MacBook]$ ls
demo1/
[MacBook]$ cd demo1
[MacBook]$ ls
README.md
```


Files clone to local machine

We normally include a .gitignore file so we don't push unnecessary files to the remote

```
$ vi $loc/activities/day13/.gitignore #standard CS50 .gitignore
```

```
<snip>
14
15 # Object files and libraries
16 *.o
17 *.a
18 a.out
<snip>
```

.gitignore file tells Git not to track files that do not need to be tracked (normally because they can be easily recreated like *.o or executables)



```
$ cp $loc/activities/day13/.gitignore .
```

```
$ git status
```

```
On branch main
```

```
Your branch is up to date with 'origin/main'.
```

```
Untracked files:
```

```
(use "git add <file>..." to include in what will be committed)
```

```
.gitignore
```

```
nothing added to commit but untracked files present (use "git add" to track)
```

```
$ git add .gitignore
```

```
$ git commit -m "added gitignore"
```

```
[main cc9defd] added gitignore
```

```
1 file changed, 38 insertions(+)
```

```
create mode 100644 .gitignore
```

```
$ git push
```

```
Enumerating objects: 4, done.
```

```
Counting objects: 100% (4/4), done.
```

```
Delta compression using up to 40 threads
```

```
Compressing objects: 100% (3/3), done.
```

```
Writing objects: 100% (3/3), 715 bytes | 65.00 KiB/s, done.
```

```
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
```

```
To github.com:cs50-2022-fall/demo1.git
```

```
2aec1ce..cc9defd main -> main
```

Copy .gitignore into Git directory, add, commit, and push

Agenda

1. Git vs GitHub
2. Working with a repo
-  3. Branches
4. Activity

Branches allow you to work on a branch without changing other branches

git-school.github.io/visualizing-git/

Google Sch

Free Explore

```
Have fun!  
$ git branch -m main  
* master  
$ git commit  
$ git branch temp  
$ git checkout temp  
$ git commit  
$ git commit  
$ git checkout master  
$ git commit  
$ git merge temp  
$ git commit
```

Local Repository
HEAD: master

Work done on a branch does not change other branches

Typically create a branch for a new feature and switch to that branch

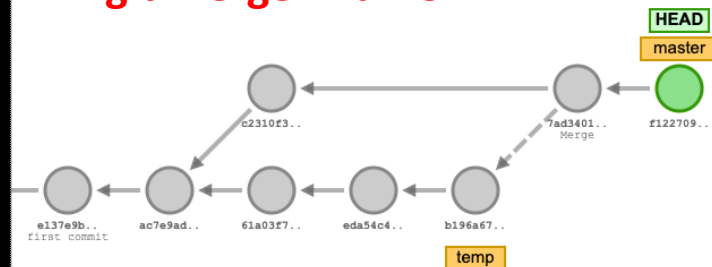
- **git branch <name>**
- **git switch <name>** (old style was **git checkout <name>**)

Work on feature until debugged and working

- **git push origin <name>**

Merge back with main when done

- **git switch main**
- **git merge <name>**



NOTE: This site uses the old master branch and uses checkout instead of switch

Agenda

1. Git vs GitHub
2. Working with a repo
3. Branches
-  4. Activity

