GIT - part one

Presented by Owen and Sam

Obligatory Pet Tax

Managing your files

```
changhem@unix8:~/private/homework$ ls
hw1-backup.py      hw1-oh-no-the-bug-appeared-again.py
hw1-bug-fixed.py   hw1-please-work.py
hw1-bug.py         hw1.py
hw1-copy.py        hw1-really-final-this-time.py
hw1-final.py
```

The Problems

- It's clunky
  - Too many files
- You have to remember which file corresponds to which version
  - You might need really long file names
- Unmanageable for increasing number of files
- What if I want to use some code for one of the files and some from another one?

Software Development

- Imagine that you're working on an operating system: Windows, MacOS, Android, iOS, Linux, etc.
  - That is a lot of code..
- 1000s of developers all working on different features, bug fixes, performance improvements
- You need to have a really good way to track, integrate, and deal with everyone's changes
  - Someone might break the system
- How does this software get developed?

What is Git?

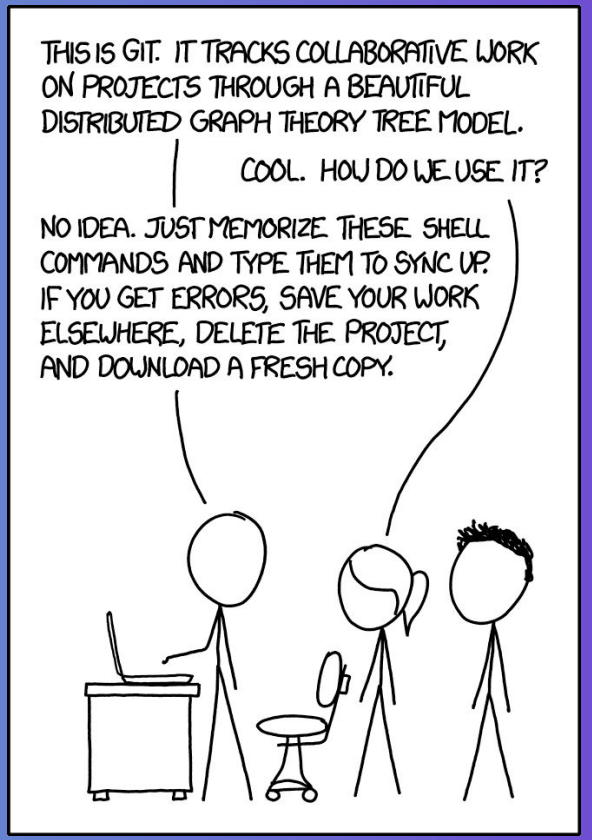
- Git is a beautiful version control system (command-line tool)
- It is quite literally a time machine for your code
- Allows you to really easily work with other people
  - Automatic tools to merge changes
  - Efficient algorithm to show you the difference between files
- It stores a your project under a magic folder which we call a repository

What is Git?

- And it gives us some crazy powerful tools to do version control and so much more!!
  - You don't have to know every single command
- Think of it as Google Docs but for your code!!
- git != Github
  - Github is a company that allows you to upload your git repos to the cloud
  - It also lets you share your code really easily
  - It also offers a ton of tools on top of git for developers (Issues, Wiki, PRs, Actions)

Our git repository

- Haven't we been using git for our GPI labs?
- Yes
- Simple commands
- There is so much more to it



Getting started with git

MacOS

Comes with git!

Linux

Ubuntu: sudo apt install git

Other distros: package manager

Windows

You can download it here:

https://git-scm.com/book/en/v2/Getting-Started-Installing-Git

Andrew Server

Already installed! No need for setup for this lab

Getting started with git

- Let's say you're starting a project
  - You want to add version control to your project so that you can track its state at different stages and potentially go back to one of them if you make breaking changes
- Let's turn your project into a git repository.
- In the root folder, run this command
  - `git init`
- What just happened?
  - Your project is now being managed by git
  - You will be able to find a .git/ folder, which contains the information that git uses, including the snapshots of your project
- This is what your git graph looks like right now.

Checking the status

- You already have some files for this project
- What is git doing with that?
- We can take a look by using:
  - `git status`
- Two things to see here
  - No commits yet
    - What are commits?
  - Untracked files
    - What are those?
    - Do we want to track our code?

Tracking files

- You want to track files that are important to you
  - Like the code that you worked really hard to write
  - Git will only care about files that you track
  - Tracked files will be later saved in a commit
- There are also files you don't want git to track
  - compiled files, log files, editor config, etc.
- Git will not track anything unless you tell it to!
- To tell git to track a file:
  - `git add [path to file or folder]`
- If you want to add all changes/files in the current folder, we often use:
  - `git add .`
- What if you want to add all changes/files except one or two?
  - .gitignore

Note: tracking your files isn't permanent. Once you commit your changes, you might have to track them again.

Committing your changes

- A commit is a bunch of changes that you want to save and take a snapshot of
- Later, you will be able to jump around between these snapshots
- You can also write a message describing your changes
- You can commit by running:
  - `git commit`
    - This will open your text editor (vim by default) to write a message
  - `git commit -m "your message"`
- A commit will be a node in the git graph

Git work cycle

- We can keep making new changes and committing them
- Make a new file
  - `touch we-love-gpi`
- git status
  - What do you see?
- We need to add we-love-gpi
  - `git add we-love-gpi`
- We need to commit these changes again
  - `git commit -m “we love gpi”`

Git work cycle

- What is the process we just did?

1. Make some changes
2. Stage the changes with git add
3. Commit the changes with git commit
    a. This gets added to your git graph
4. Rinse and repeat!!

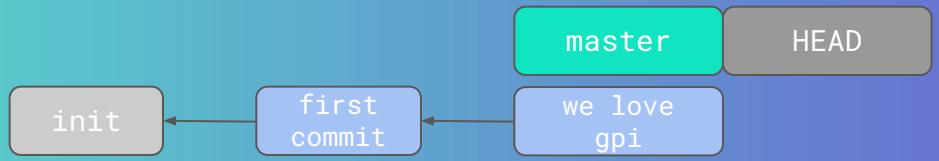
Where is the graph?

- To visualize the repository's git history as a graph
  - `git log --graph --decorate`
  - type "q" to quit the visualization
- Notice: each commit has a "hash" - like a unique identifier
  - Commit hash is useful for some commands - we will come across later
- Also notice: "where you are" on the graph is pointed to by "HEAD"
  - This refers to the current version of your repository
- But! This is the most boring graph - a straight line

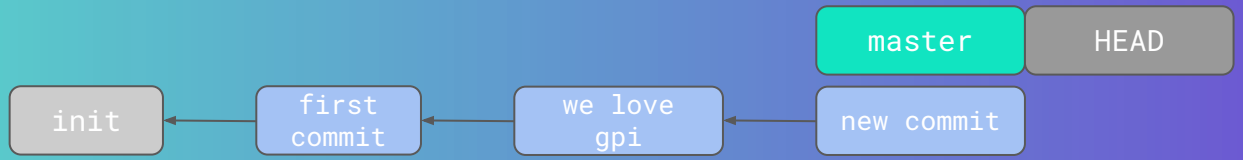
Branching and the git commit tree

- So far, all our work has been on a branch named “master”
  - This is usually the default branch
- A branch is a separate chain of commit history
- A branch is just a pointer to some specific commit
- “HEAD” is also a pointer to, usually a branch, but possibly a commit
- Useful resource: https://learngitbranching.js.org/

What happens when you commit?



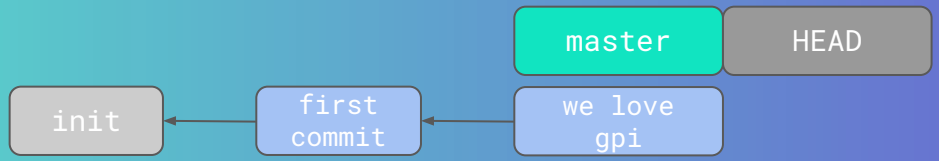

```
git commit -m ‘new commit’
```



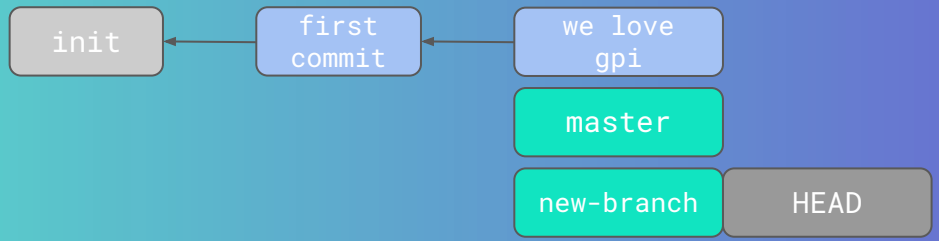
Working on another branch

- To make a new branch
  - `git branch <new-branch-name>`
  - Note: this command does NOT switch to the new branch
- Switch between existing branches
  - `git checkout <branch-name>`
- To make a new branch AND switch to it
  - `git checkout -b <new-branch-name>`

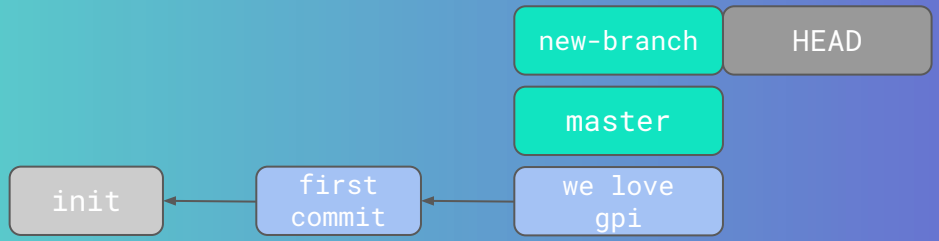
What happens when you create a new branch?




```
git branch new-branch; git checkout new-branch
```

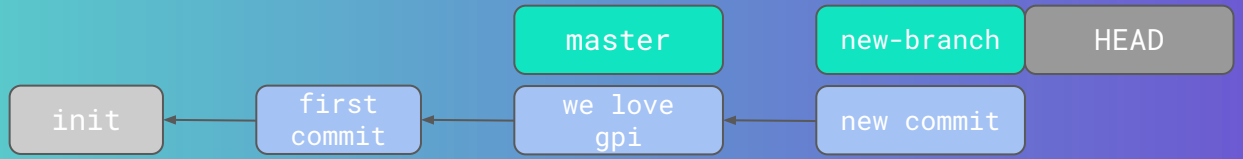


What happens when you commit on another branch?

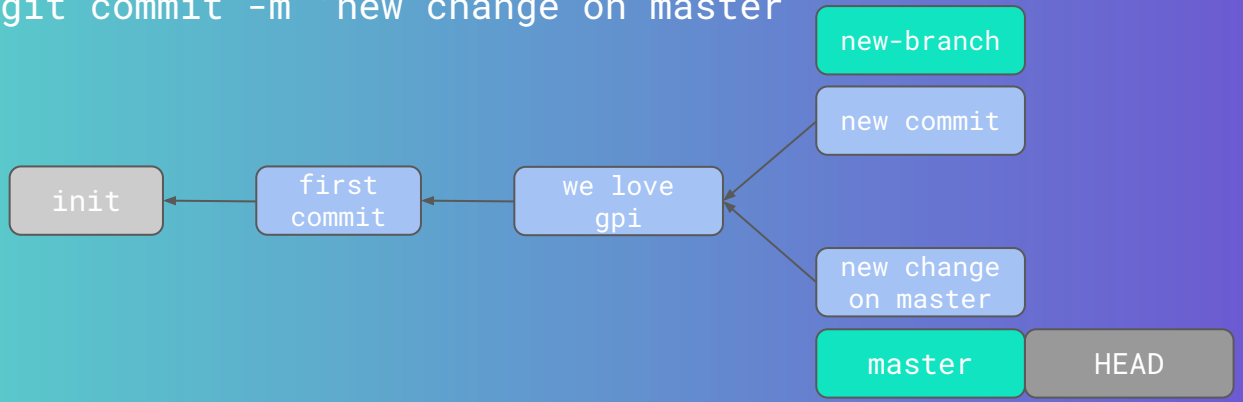



```
git commit -m 'new commit'
```

What happens when you commit on another branch? (continue)



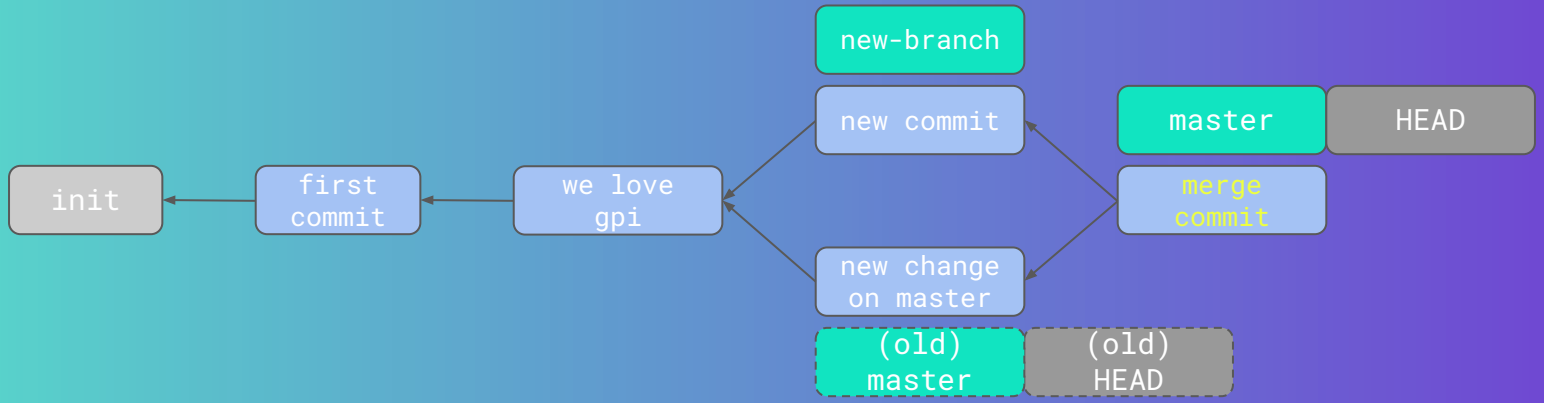

```
git checkout master
# some new changes to repo
git add .
git commit -m ‘new change on master’
```



Why is branching useful?

- Multiple people can work on different features at the same time
  - Even change the same files at the same time
  - Changes do not interfere as long as on different branches
- Can easily combine these changes - git magic
  - When HEAD is on “master”
  - `git merge new-branch`
  - Result
    - changes in “new-branch” is merged into “master”
    - “new-branch” doesn’t change

The merge commit



Merge is not always successful

- git is smart - usually automatically figures out how to merge changes on different branches
- But, sometimes you might run into “merge conflict”
  - When git doesn’t know how to merge some changes to the same file
- When there is a merge conflict - git marks the conflicting sections in the file

```
<<<<<<< HEAD
# anything here are changes on HEAD branch
=======
# anything here are changes from another-branch
>>>>>>> another-branch
```

Merge is not always successful (continue)

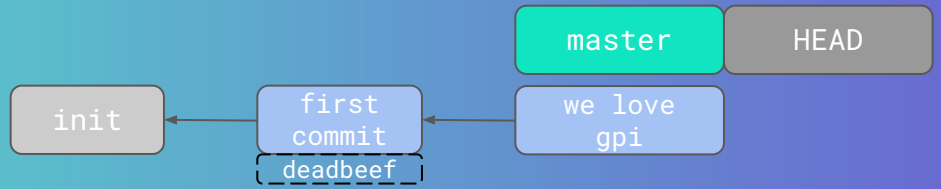
- Now, you need to “resolve” these conflicts:
  - edit the marked sections of the file - i.e. manually merge these sections
  - then remove the markings
  - `git add <file>`
  - once you are done with all conflicts in the repo
  - `git commit` to finish merging

```
<<<<<<< HEAD
# anything here are changes on HEAD branch
=======
# anything here are changes from another-branch
>>>>>>> another-branch
```

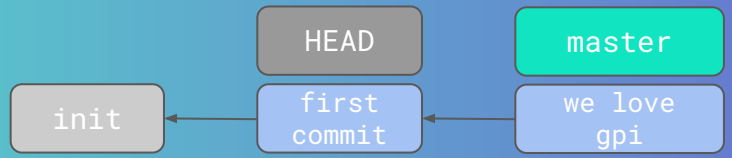
Final caveat - detached HEAD

- `git checkout` moves between branches
  - but you can also `git checkout <some-commit-hash>`
- If the commit is not the head (most recent) commit of any branch - detached HEAD state

detached HEAD




```
git checkout deadbeef
```



detached HEAD (continue)

- In detached HEAD state
  - You can look around - see this version of the repo
  - Edits will be lost once you jump away
- What if you want to make changes from this commit? - create a new branch!

Hints for the lab

- Before you start, run ./setup.sh
- This week's lab directory is a git repo by itself, despite being inside the gpi-labs repo
- When you are doing the labs, run the commands inside this week's lab directory
- When you are done with the lab, commit your changes under gpi-labs directory
- Don't forget to commit and run driver between stages!
- `git branch` shows you what branches exist and which one HEAD is on
- `git revert <commit-hash>` reverts a commit
  - by making a new commit with opposite changes
- Always run the driver to make sure you are on the right track!