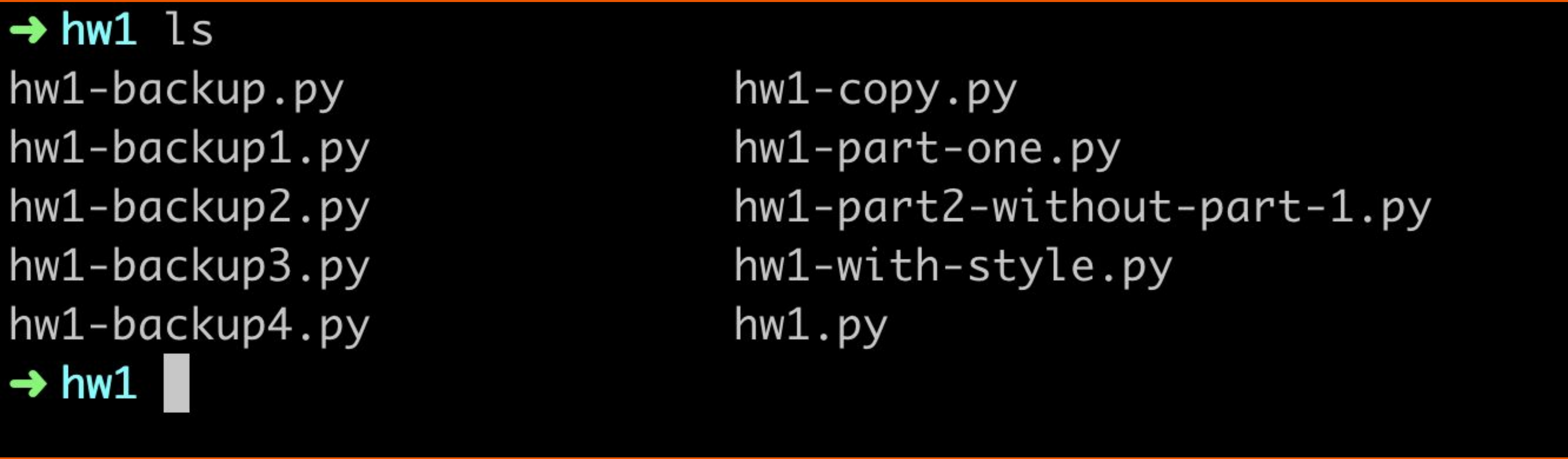
Git: Part I

Emmanuel Eppinger

Dogo Tax

Dogo Tax

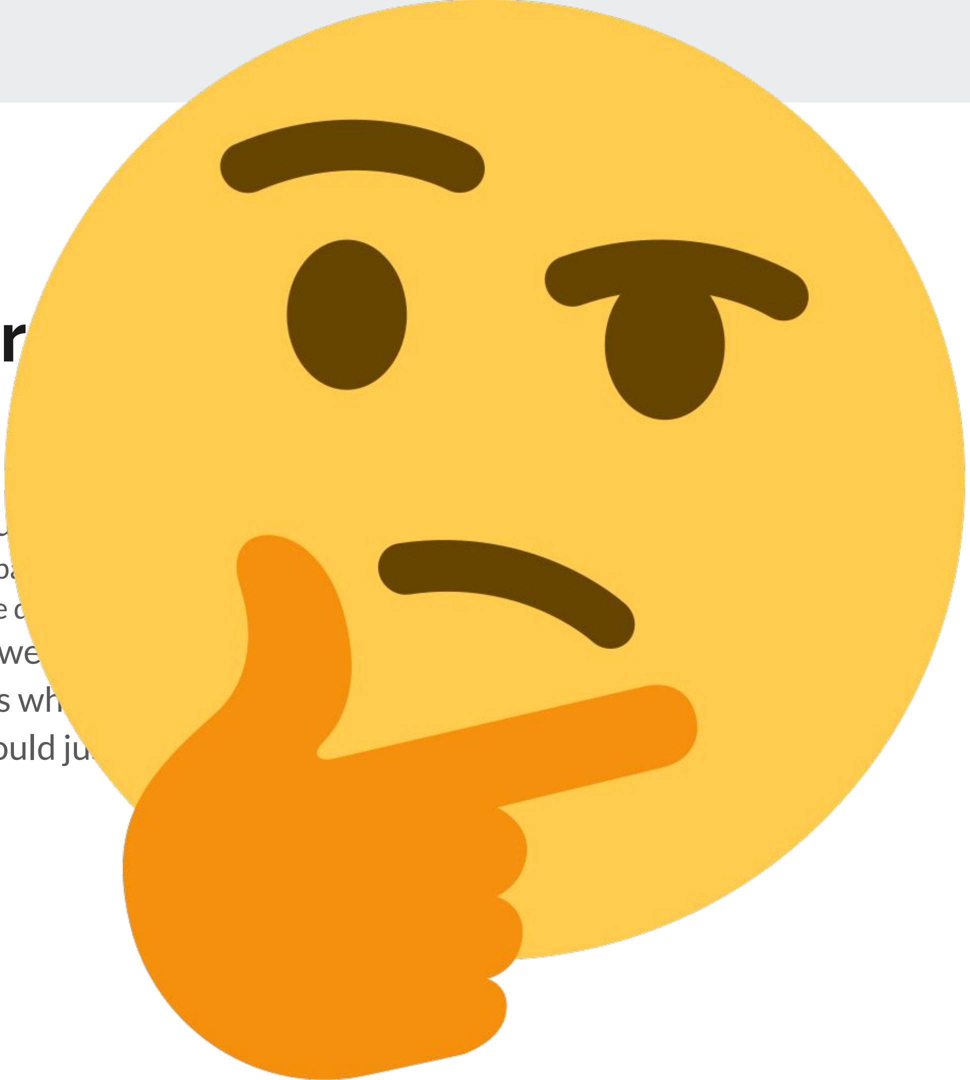
Dogo Tax



```
➜ hw1 ls
hw1-backup.py                 hw1-copy.py
hw1-backup1.py                hw1-part-one.py
hw1-backup2.py                hw1-part2-without-part-1.py
hw1-backup3.py                hw1-with-style.py
hw1-backup4.py                hw1.py
➜ hw1 
```

What's wr

- Its clunky
- It relies on you
    - Ex: how pa
    - What the c
- Switching betwe
- What happens wh
- What if you could ju



But also developing software is complicated

- Imagine you’re working on an operating system: windows, macos, android, ios, linux, etc
    - It’s a lot of code
- 1000s of developers all working on different features, bug fixes, performance improvements
- You really need to have a really good way to track, integrate, and deal with everyone’s changes
- How does this software get developed?

What is git?

- Git is a beautiful version control system
- It is quite literally a time machine for your code
- Allows you to really easily work with other people
- It stores a your project in a magic *folder* which we call a *repository*
- And it gives us some crazy powerful tools to do version control and so much more!!
- Think of it as Google Docs but for your code!!
- git != github
  - Github is a company that lets you use git to backup your code 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

Haven't we'v n all the HWs?

- Yes
- All of you may have se
- But...
- There is actually a lot

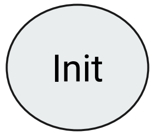
THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.
COOL. HOW DO WE USE IT?
NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.

Getting started with git

- Installing git is stupid easy
- Check if its installed with
  - $ which git
  - If it gives you a path you've already got it!!
- On mac:
  - $ brew install git
  - You should install homebrew if you haven't yet (brew.sh) <- will literally save your life
- On linux and WSL
  - $ sudo apt-get install git
- On andrew:
  - Nothing!! It's already installed so you don't need to install anything for the HW

Getting started using git

- Let’s say you’re starting a project
    - You’ve maybe written a few files and it’s really starting to come together
    - You want to start being able to save your progress and take snapshots of the project at different stages
- In the folder for your project run
    - $ git init
- What just happened?
    - We told git that to make a new git repository in this folder
- This starts the first node on our graph



Checking what git is doing?

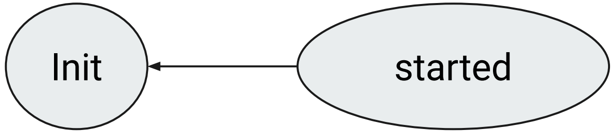
- You had some code that you already wrote for this project
- What is git doing with that?
- We can check what git is doing by using:
  - $ git status
- Two things to see here
  - No commits yet
    - What are commits?
  - Untracked files
    - What are those?
    - Do we want to track them and why?

Tracking files

- Sometimes you have files you want git to keep track of
    - Usually the code you work really hard to write
- There are also files you don't want git to track
    - Compiled files, log files, etc.
- Git won't track anything that you haven't told it to
- Git won't track any changes unless you tell it to
- Tell git to track a file or git it to track some changes:
    - $ git add *[path to changed file]*

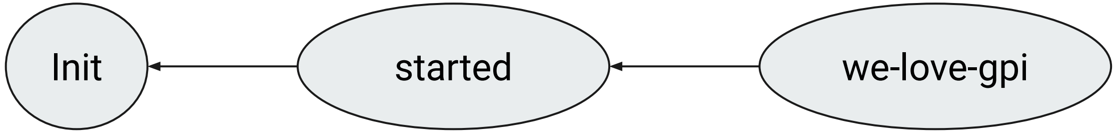
Commits: what are those?

- Commits are a collection of changes that get added to the graph
- These are the snapshots that you are able to jump between
- You also get to write a message describing what the changes you made were
- You can commit by running:
    - $ git commit
        - Will open in some text editor (vim by default) to write message
    - $ git commit -m “your message here”
        - Doesn’t open up anything



You can keep doing this as you make changes

- Make a new file
  - $ touch we-love-gpi
- $ git status
- What do you see?
- What happens when we run this command:
  - $ git diff
  - What about if we write some stuff into we-love-gpi
- We need to add we-love-gpi
- We need to commit these changes again

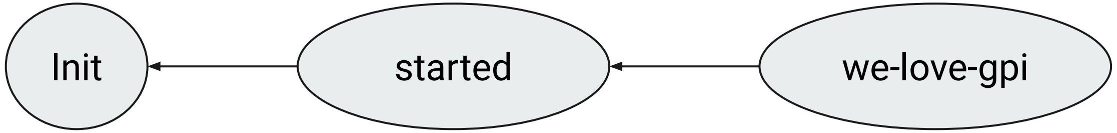


What’s the process we just did?

1. Make some changes
2. Stage those changes with git add
    a. This moves your changes into what is called the staging area
3. Commit those changes with git commit
    a. Commits your changes onto the tree
4. Repeat and have great snapshots of your work!!

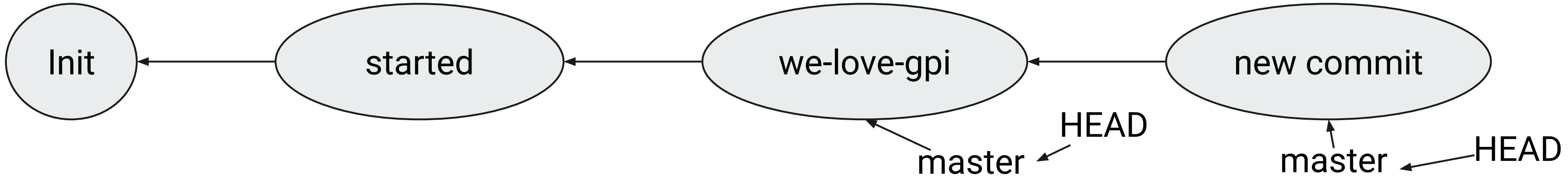
There is this tree thingy, how do i see it?

- Great question!!
- You can use a command
    - $ git log --graph --decorate
    - You can get out of git log by pressing “q”
- You can see you’re entire commit history all the way back to the git init
- Do you notice the git hashes?
    - They look like this: 06a12a2465b78ca92f08aacf774cb98fda3c3519
    - They will be useful later



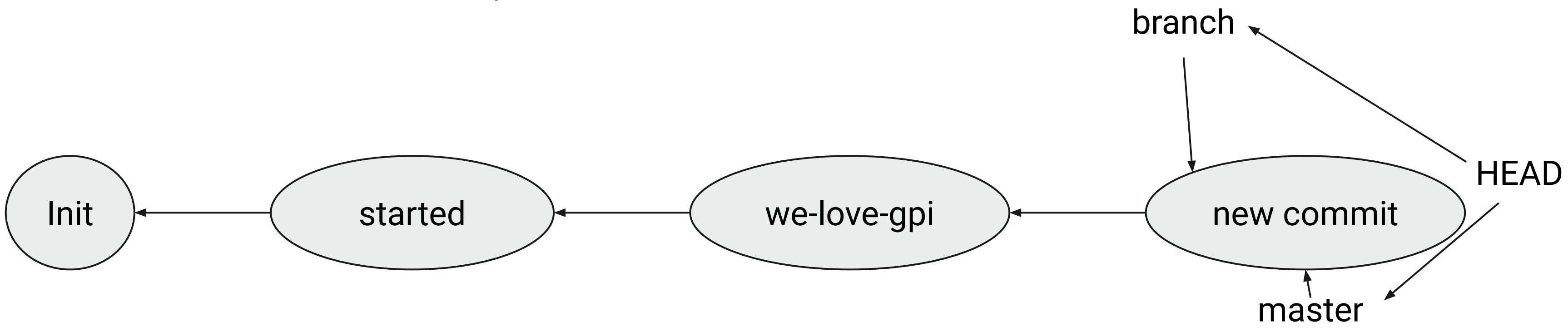
Aren’t all these trees straight lines?

- Yes
- But you can change that by giving your trees branches
- So by default there is one branch called master
  - When you commit you extend the branch you’re currently on
- You keep track of where you currently are on the tree with the HEAD
  - When you commit you move what your current branch points to and therefore move the HEAD
  - HEAD always points to a branch



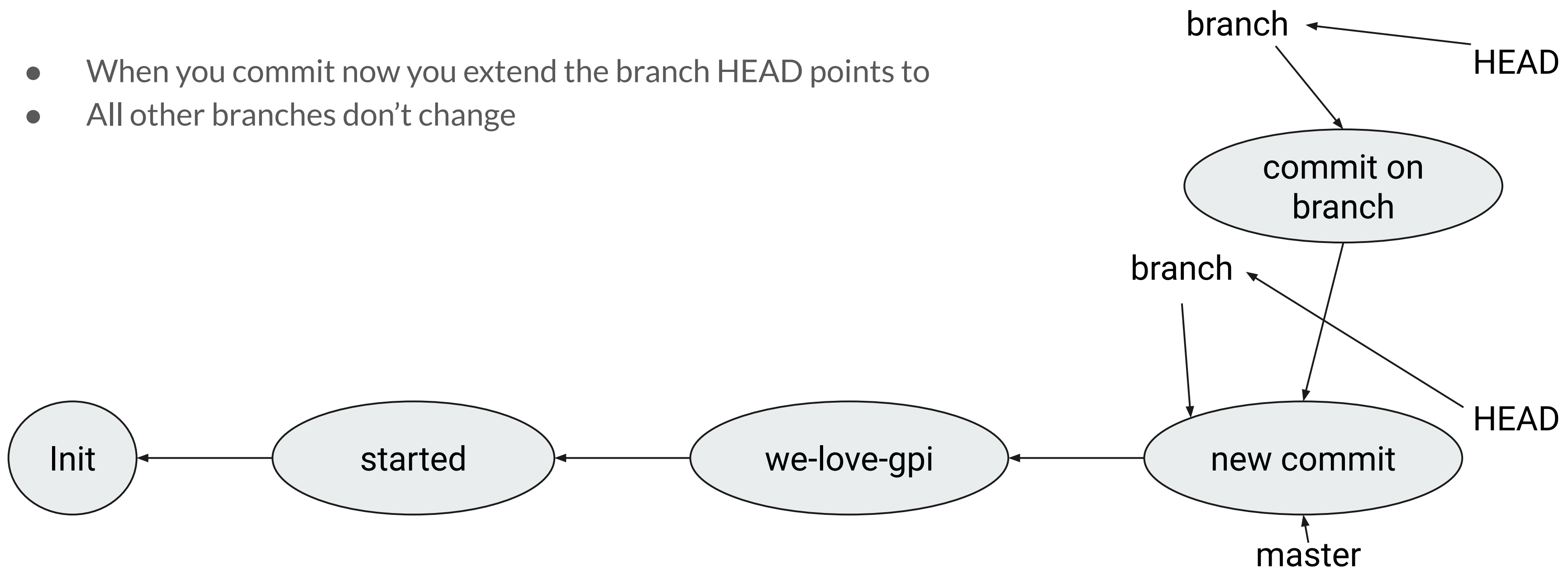
Making branches

- You can make a branch from a commit with
  - $ git branch [branch name]
  - $ git checkout [branch name]
- *branch* makes a new branch
- *checkout* switches the head to point to that branch



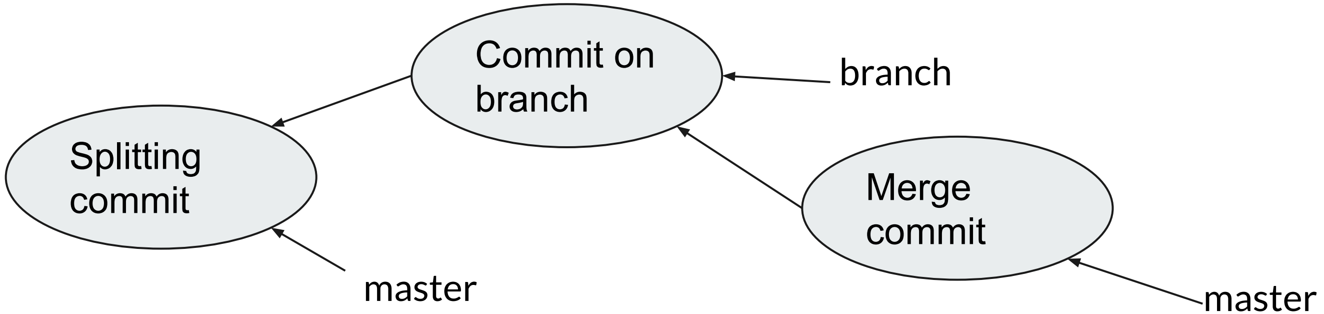
Making branches

- When you commit now you extend the branch HEAD points to
- All other branches don't change



Combining branches

- Really nice thing about branches is you can have multiple people working on different parts of the project at the same time
- They can do this without breaking each other’s versions of the project
- When they want to combine two branches you can:
  - $ git merge [branch you want to merge]
- This makes a commit that both branches and HEAD point to

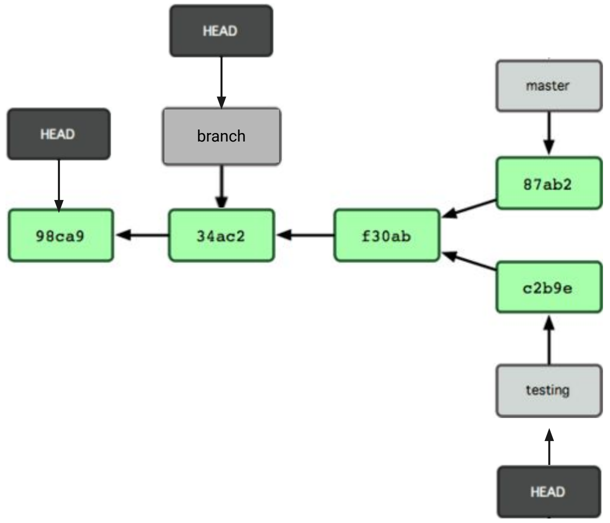


How to actually merge branches?

- Merge the branches
  - $ git merge [branch name]
- Check to see if there were any issues merging
  - $ git status
- Fix all the conflicts
  - If there is a conflict in a file, git will surround the section that needs to be fixed with >>>>>>> or <<<<<<<
  - You then need to combine those sections to finish the merge
- Stage and commit your changes
  - $ git add file1 file2 file3 …
  - $ git commit
- Yay you merged two branches together

When do we get to time travel?

- Right Now!!
- To jump between commits you can use:
  - $ git checkout [branch name]
- Use this to jump around between branches!!
- What do you think this is doing with HEAD?
- You can also checkout a commit with
  - $ git checkout [commit hash]
  - What branch are you on now?
  - You’re not, you have a detached head



How to deal with a detached head?

- Go to the hospital
- You can make a new branch when you checkout the commit
  - $ git checkout -b [branch name]
- If you are confused by branching, there are interactive visualizations of branching and merging at:
  - learngitbranching.js.org

Helpful hints for the lab

- For the lab, this week's lab is a git repo, but all of our labs are in one git repo
- Git is smart enough to look for the closest .git folder so do the lab in this labs folder, and then commit all your changes from the gpi-labs folder
- Don't forget to commit and run driver between stages!
- Remember that git branch will show you what branch you're on and which branches exist
- Switch between branches using git checkout
- You can list your branches using
  - $ git branch l
- To revert to a commit:
  - $ git revert [commit hash]
- Always run the driver to make sure you're on the right track!!!