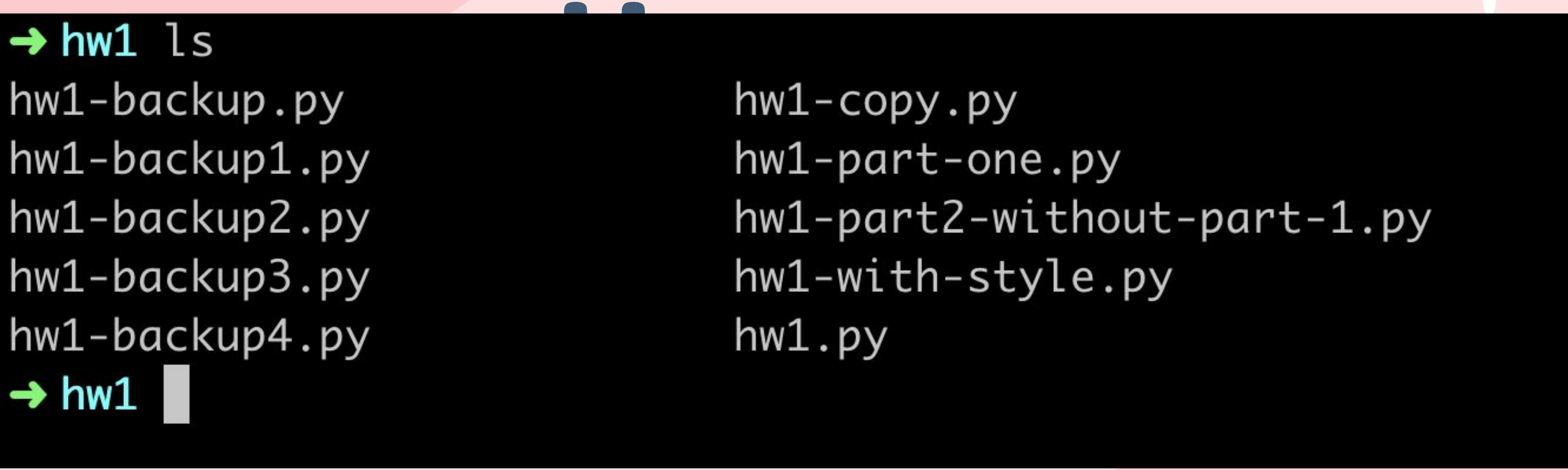
GIT : [gɪt] (part 1)

Jessica Dai & Kyle Booker

Pet Tax

Announcements

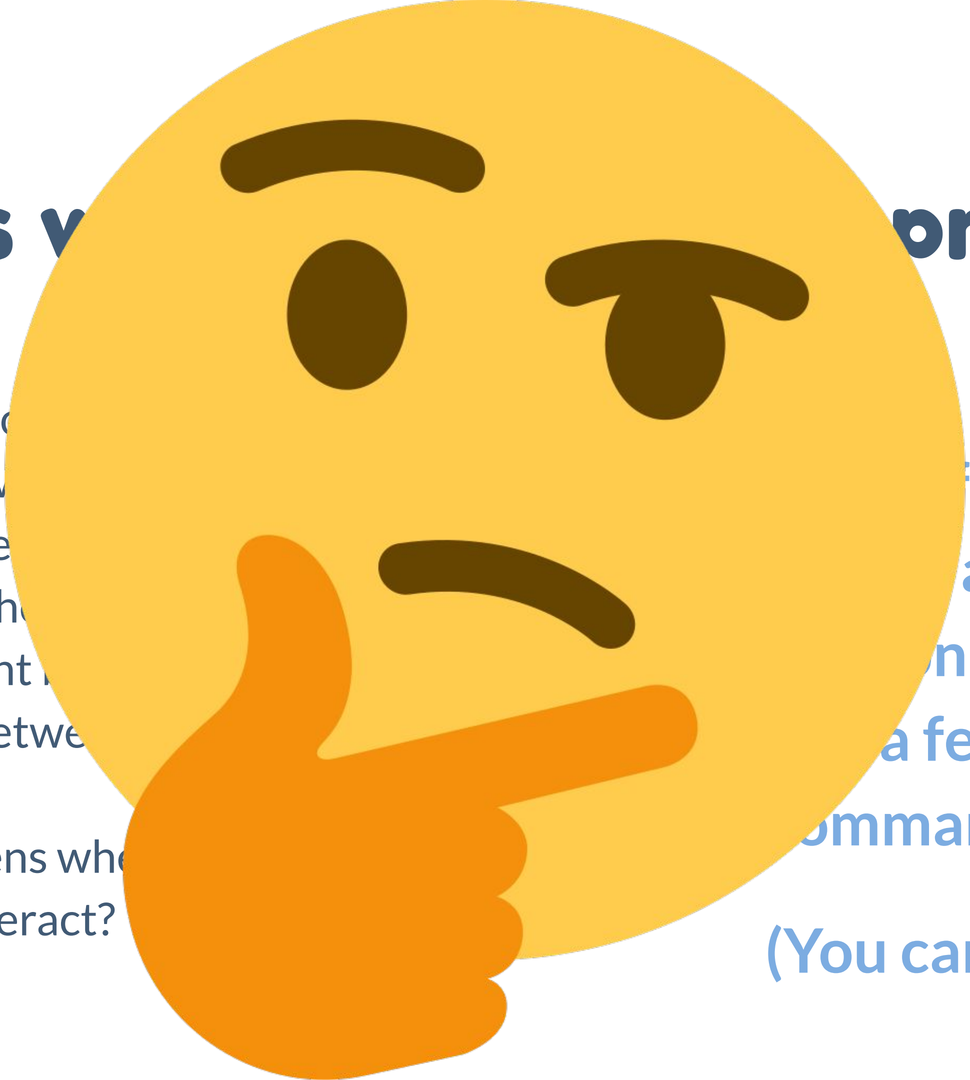
- Midterm 1 will take place **during class on Oct 5 (Section A)/Oct 7 (Section B)**
- You will have the full 80 minutes to take the exam. Please try to arrive early so we can start on time. Do not stress about the test! The exam will be heavily curved. :)
- The exam will cover everything we have learned in class up to Git part 1. The test format will include multiple choice and short answer and it will be conducted using pen/pencil on paper. Man pages will be provided.
- You can bring one double-sided handwritten page of cheatsheet, letter-sized (8.5 x 11). You are allowed to write it digitally, but it must be printed out before the exam. Please place your name at the top of the cheatsheet as we will be collecting your cheatsheets at the end.
- Regardless of how many late days you have left, all labs must be turned in by **the hard deadline of October 23rd.** After this date, unsubmitted labs will be marked as a 0.



```
➜ 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 is v proach?

- Its clunky
- It relies on yo
  - Ex: how togethe
  - What th different
- Switching betwe and pasting
- What happens whe files that interact?



f you could all that n control in a few mmands?

(You can)

Developing software

1. Imagine you’re working on an operating system: windows, macos, android, ios, linux, etc
    a. It’s a lot of code (the Linux kernel is around 27.8 million lines)
2. 1000s of developers all working on different features, bug fixes, performance improvements
3. You really need to have a really good way to track, integrate, and deal with everyone’s changes
4. How does this software get developed?

What is Git?

- It is an open source distributed version control system used for tracking changes in any set of files
- Usually used for coordinating work among programmers collaboratively developing source code during software development.
- GIT IS NOT GITHUB!
- Stores project locally! for everyone involved :)

Wait...

Haven’t w all the HWs?

1. Yes
2. All of you h
3. But…
4. There is act probably know

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.

Got Git?

- On mac:

  ```
  $ brew install git
  ```

  You should install homebrew if you haven’t yet (brew.sh) <- will literally save your life

- On linux and windows (Windows Subsystem for Linux)

  ```
  $ sudo apt-get install git
  ```

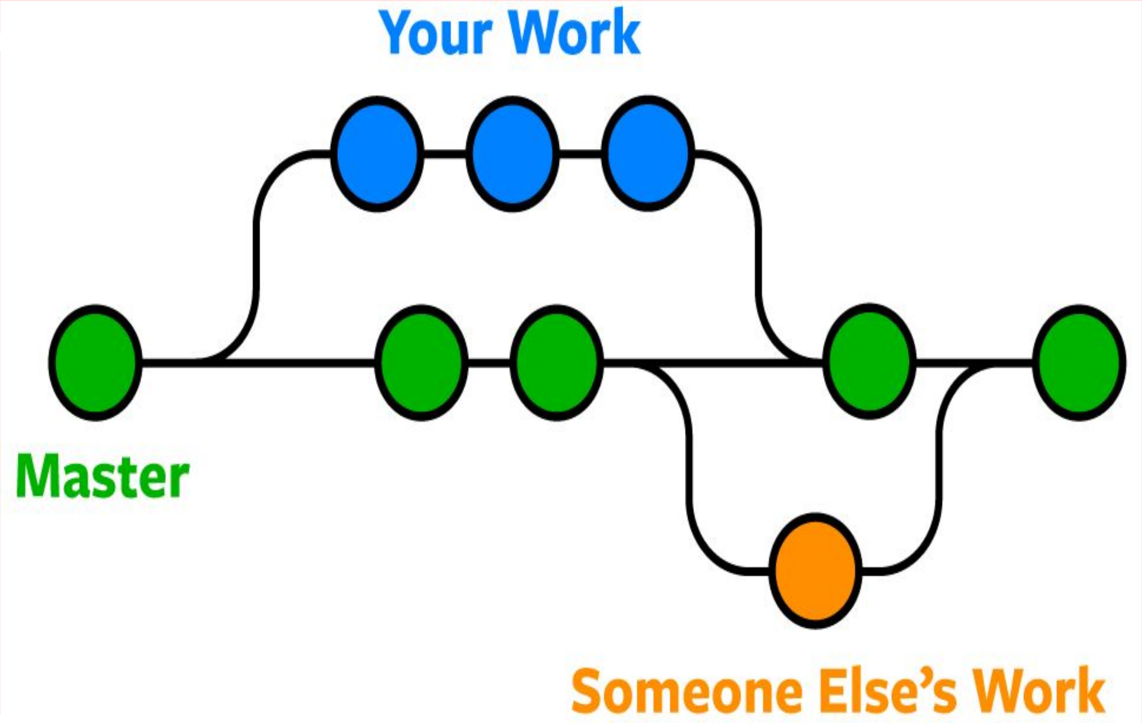
- It’s already on the andrew servers!

Check if you have git installed!

```
jdai2@linux-15:~$ which git
/usr/bin/git
```

Andrew has it installed!

How to git Git- a visualization



Git workflow!

There are many different types depending on your goals and team.

Git branching workflow: (shown)

Every feature gets its own branch when developers commit to this workflow. Developers create a branch, make changes, and then merge it into main.

Gitting started with a repo

What is a repository?

A Git repository (or repo for short) contains all of the project files and the entire revision history.

How to make a repo

Start in a project folder, then run:

```
git init
```

Created a .git sub folder that stores the entire project history
(*Remember . is hidden so run ls -a to see it!*)

git status

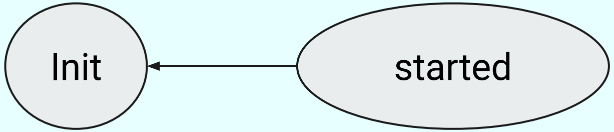
1. You had some code that you already wrote for this project
2. What is git doing with that?
3. We can check what git is doing by using:
   a. $ git status
4. Two things to see here
   a. No commits yet
      i. What are commits?
   b. Untracked files
      i. What are those?
      ii. Do we want to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

Tracking Files

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

Commits: what are those?

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



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!!

Git commands to git good!

Staging: what files do we want to commit?
$ git add [path to file or folder]
$ git add .
	We stage by using add!
	Tells git to track a file
	Tells git to track all files

Committing:
$ git commit
$ git commit -m “your message here”
	Commit files that are staged
	Prompts you to write a message (vim)
	You can write message directly!

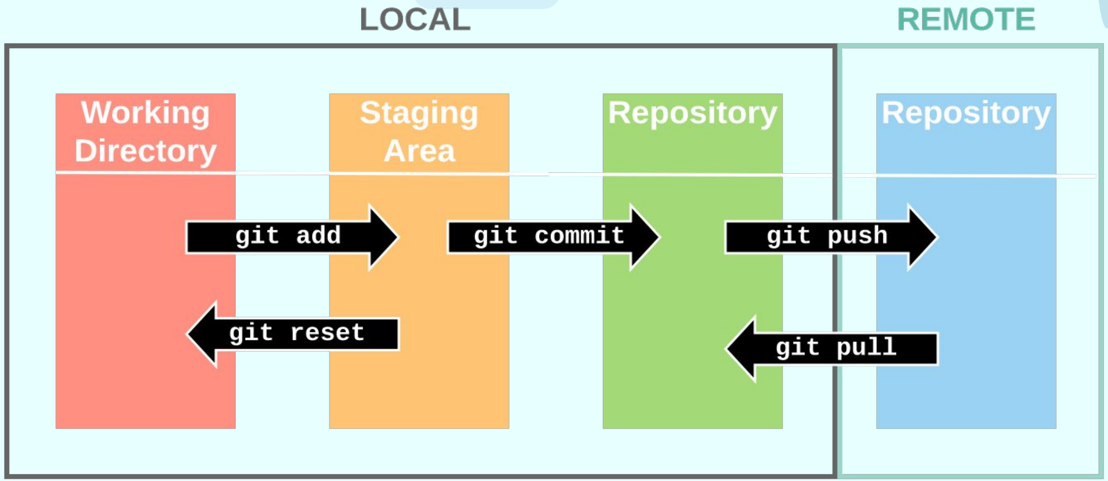
$ git status	check what git is doing with our files ● Commits ● Untracked files ● Changes not staged

Gotta Git 'Em All!

know

your

git



LOCAL

REMOTE

Working Directory

Staging Area

Repository

Repository

git add

git reset

git commit

git push

git pull

:)

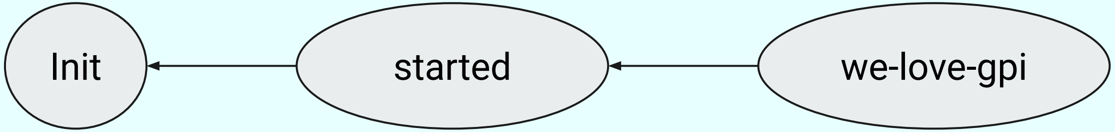
:0

:{

Questions? Don't worry about push and pull right now!

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

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



Oops! I did it again...

log

```
git log
git log --graph --decorate
```

Checks your previous commits and lists them

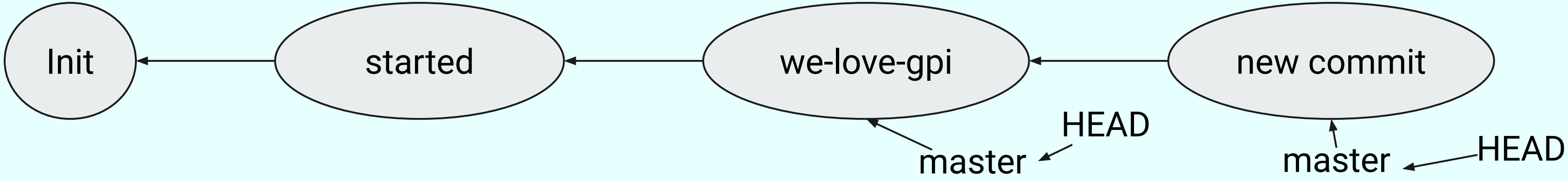
rev

```
git revert <commit-hash>
```

“Reverts a commit” by making a new commit with opposite changes!

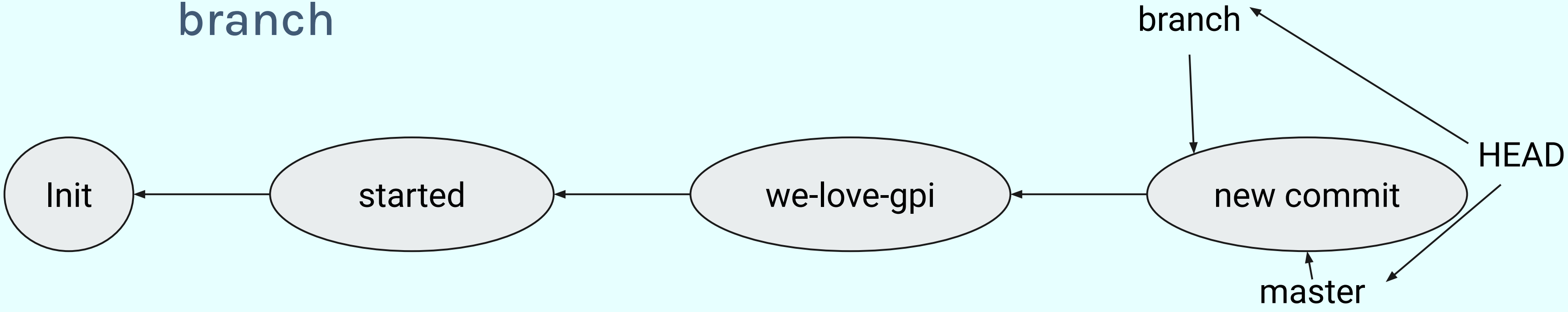
Are all these trees straight lines?

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



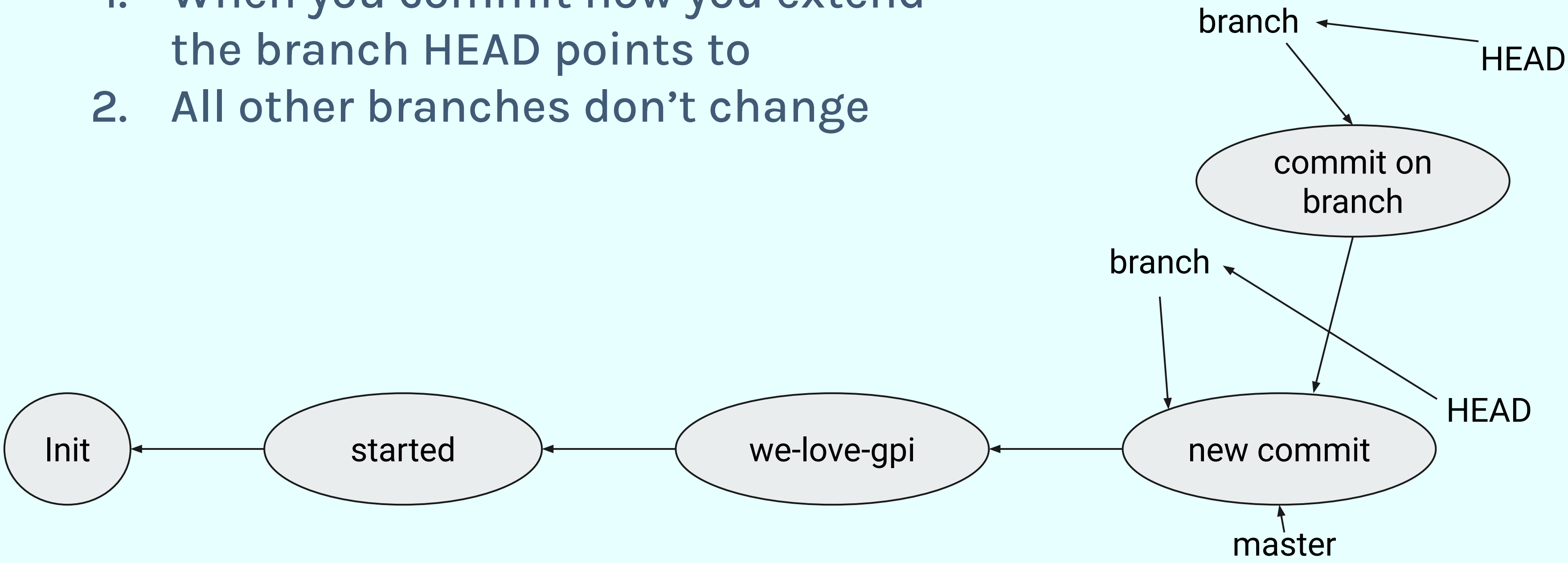
Branching out?

1. You can make a branch from a commit with
   a. $ git branch [branch name]
   b. $ git checkout [branch name]
2. branch makes a new branch
3. checkout switches the head to point to that branch



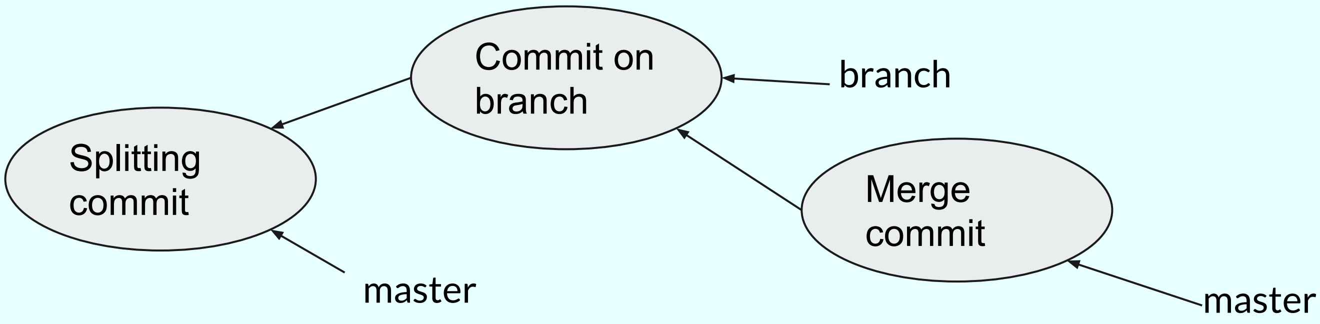
Making branches

1. When you commit now you extend the branch HEAD points to
2. All other branches don’t change



MerG it! Combining branches

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

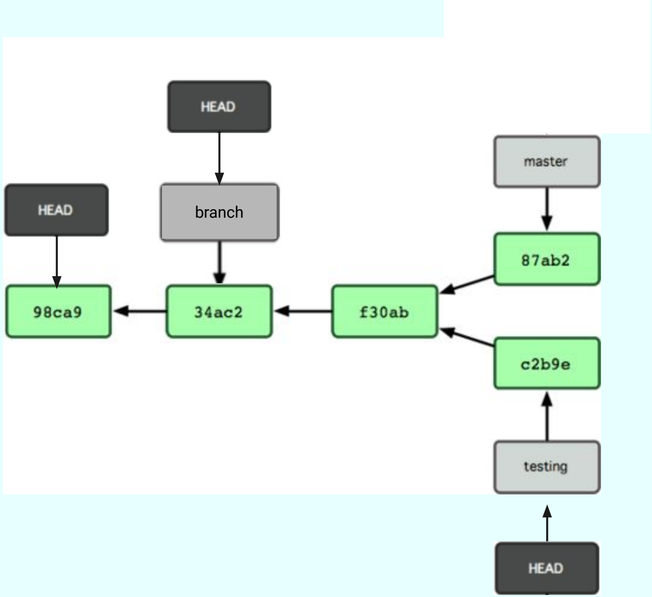


How to actually merge branches?

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

When do we get to time travel?

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



How to deal with a detached head?

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

Helpful hints

- Before you start run ./setup.sh
- 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!!!