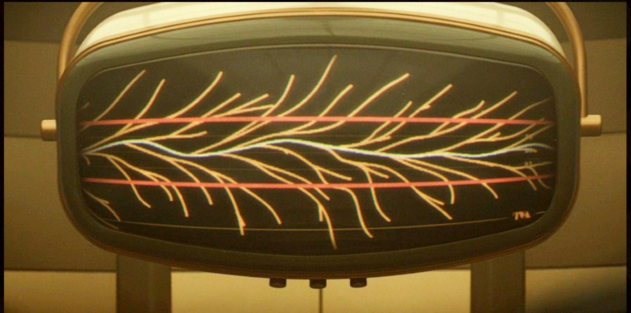
How to Time Travel

Laura, Yosef, and Keiffer



Git: Part I

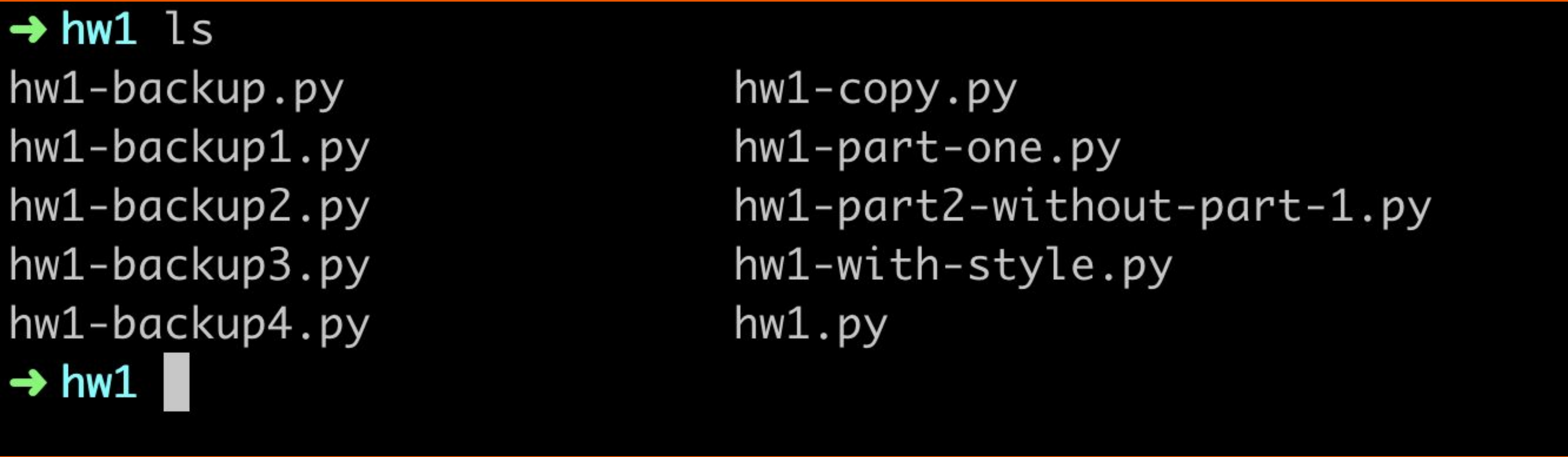
Laura, Yosef, and Keiffer

Exam 1 Logistics

- Next week during lecture time (**Oct 7th**)
- Material up until this lecture (Git Pt. 1)
- Will be multiple choice and short answer
- Paper + pen/pencil
- Materials allowed:
    - Man pages will be provided
    - 1 pg (8.5 x 11) handwritten cheat sheet (front and back)
        - You can write it out digitally but you must have it printed
        - Name at the top of the sheet
    - We will collect your cheat sheets at the end!
- It will be heavily curved so don’t worry!

Midsem Grade Logistics

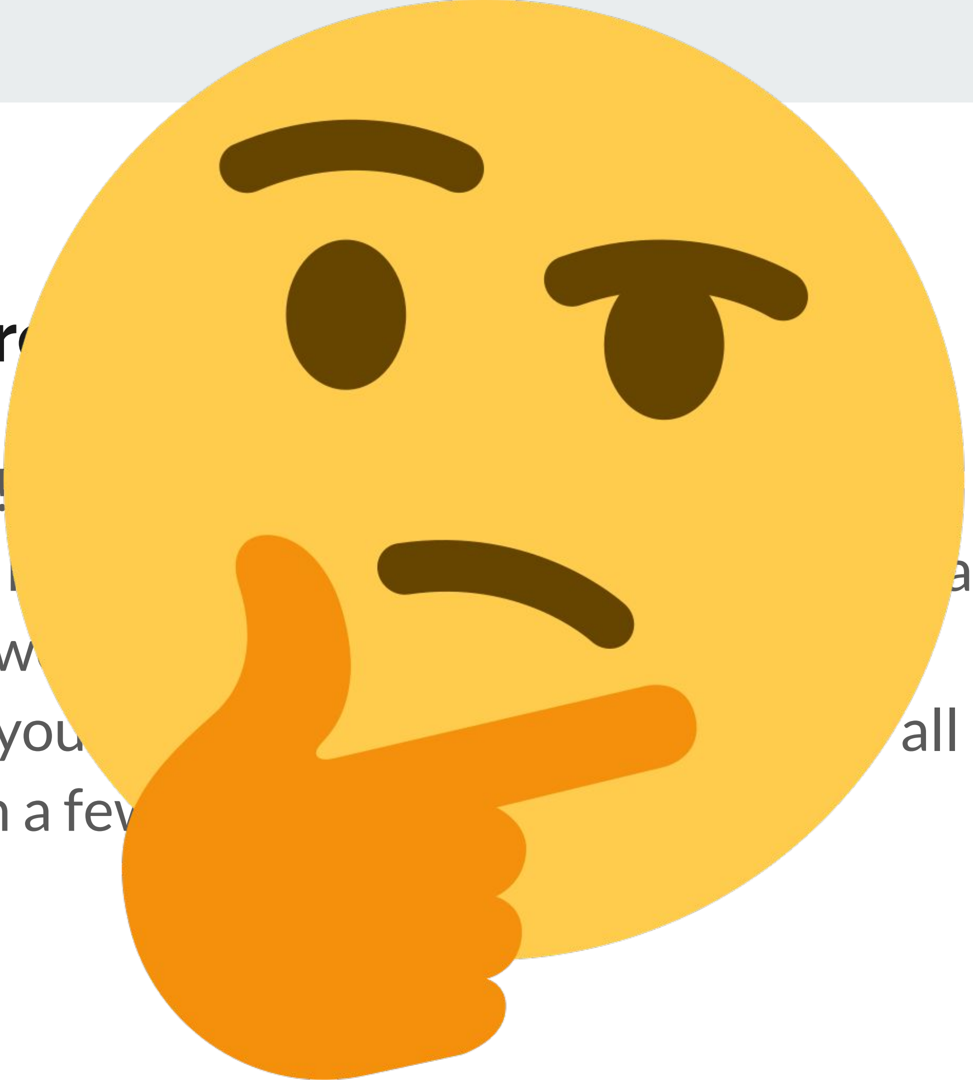
- Hard deadline for trainerlab to romancelab is **October 14th (Thursday)**
- Double check autolab!!
  - (if you didn't submit your .tex files for smashlab, you may have a negative score)
- Your midsem grade will include your midterm score



```
➜ 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

- Clunky!!
- Need to asting to restore w
- What if you all of that for you with a fev



Developing software is complicated

- Software developers everywhere use **version control** manage large projects!

What is git?

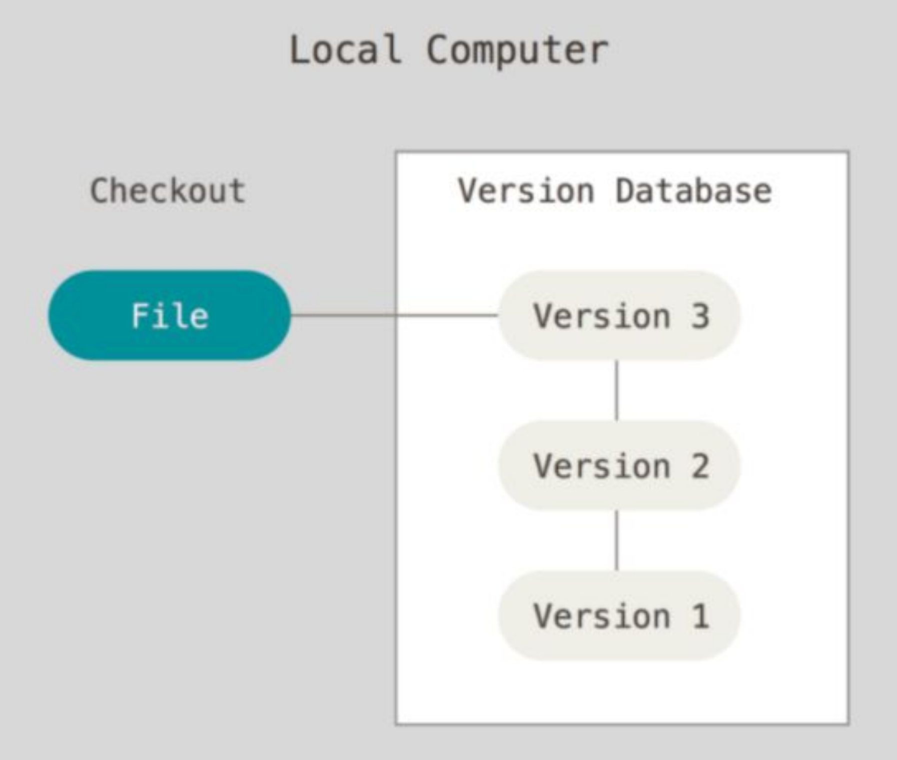
“the stupid content tracker”

- Linus Torvalds

(type “man git” in your terminal - it actually says this!)

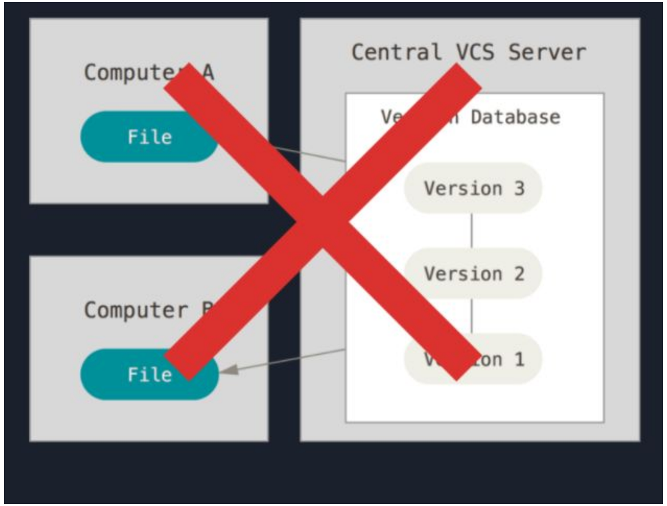
What is git?

- **Distributed** version control (work easily with other people)
- Stores the entire project **locally**
- Quite literally a time machine for your code!



What ISN'T git?

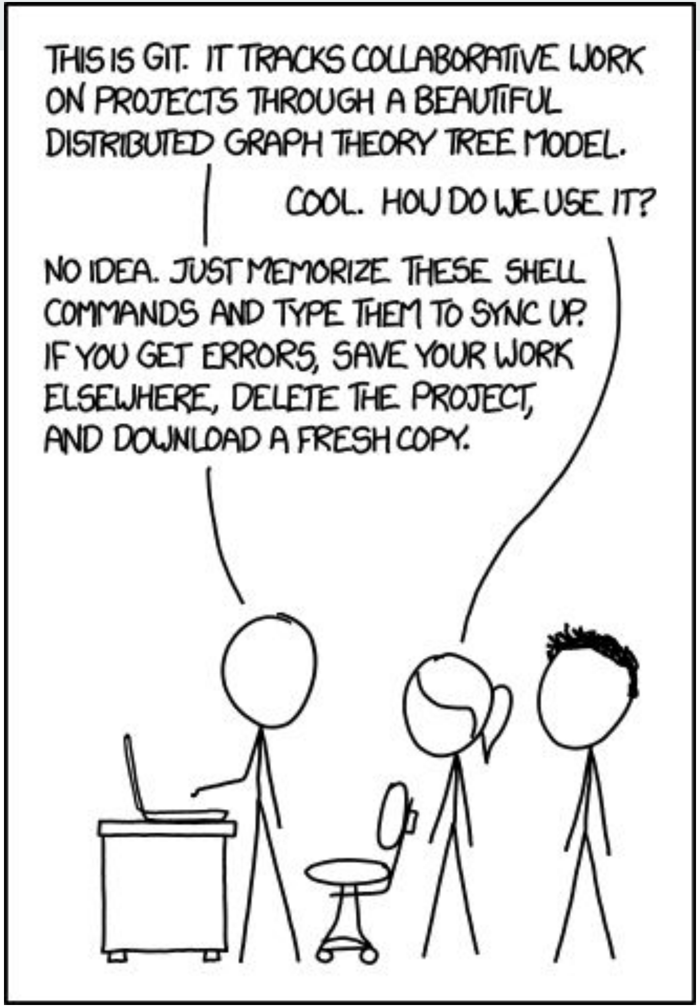
- GitHub is a website to share and collaborate on git repositories
- We'll be learning more about Github next week!



Haven’t we’ve been using git on all the HWs?

- Yes
- All of you have seen git before
- But…
- There is actually a lot more to it than you probably know
- Now you will understand why we ask you to add and commit your files

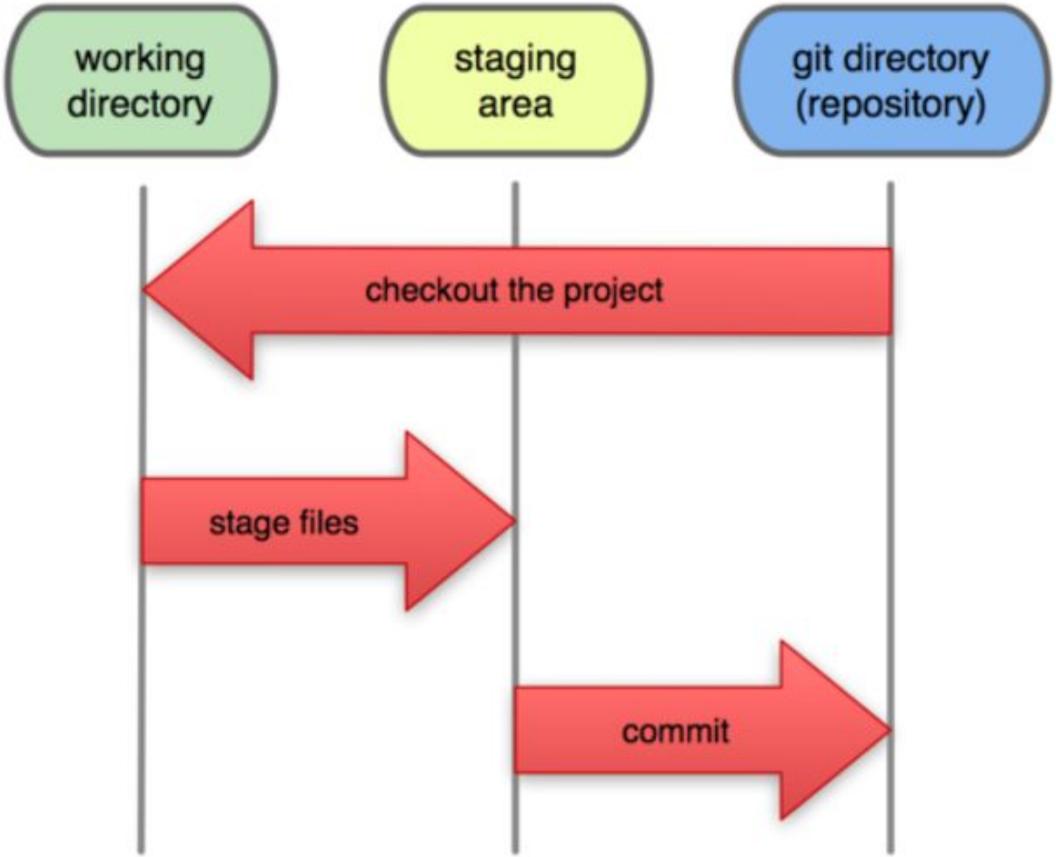
git happens....


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.

Time to git gud

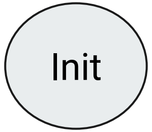
- Check if its installed with
  - $ which git
  - If it gives you a path you've already got it!!
- Mac:
  - Already installed!
- Linux:
  - Ubuntu: sudo apt install git
  - Other distros: package manager
- Windows:
  - You can download it here:
  - https://git-scm.com/book/en/v2/Getting-Started-Installing-Git
- Andrew:
  - Nothing!! It's already installed so you don't need to install anything for the HW

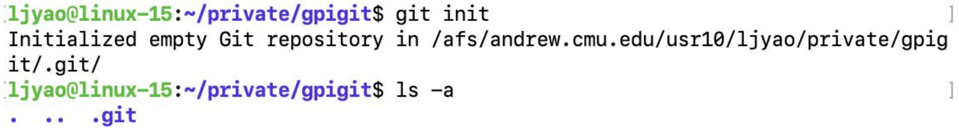
Local Operations



Getting started using git

- You started a new project with some files and want to save your progress
- Run git init in the project folder
- What just happened?
    - ○ Creates a .git folder that stores the entire project history
- This starts the first node on our graph





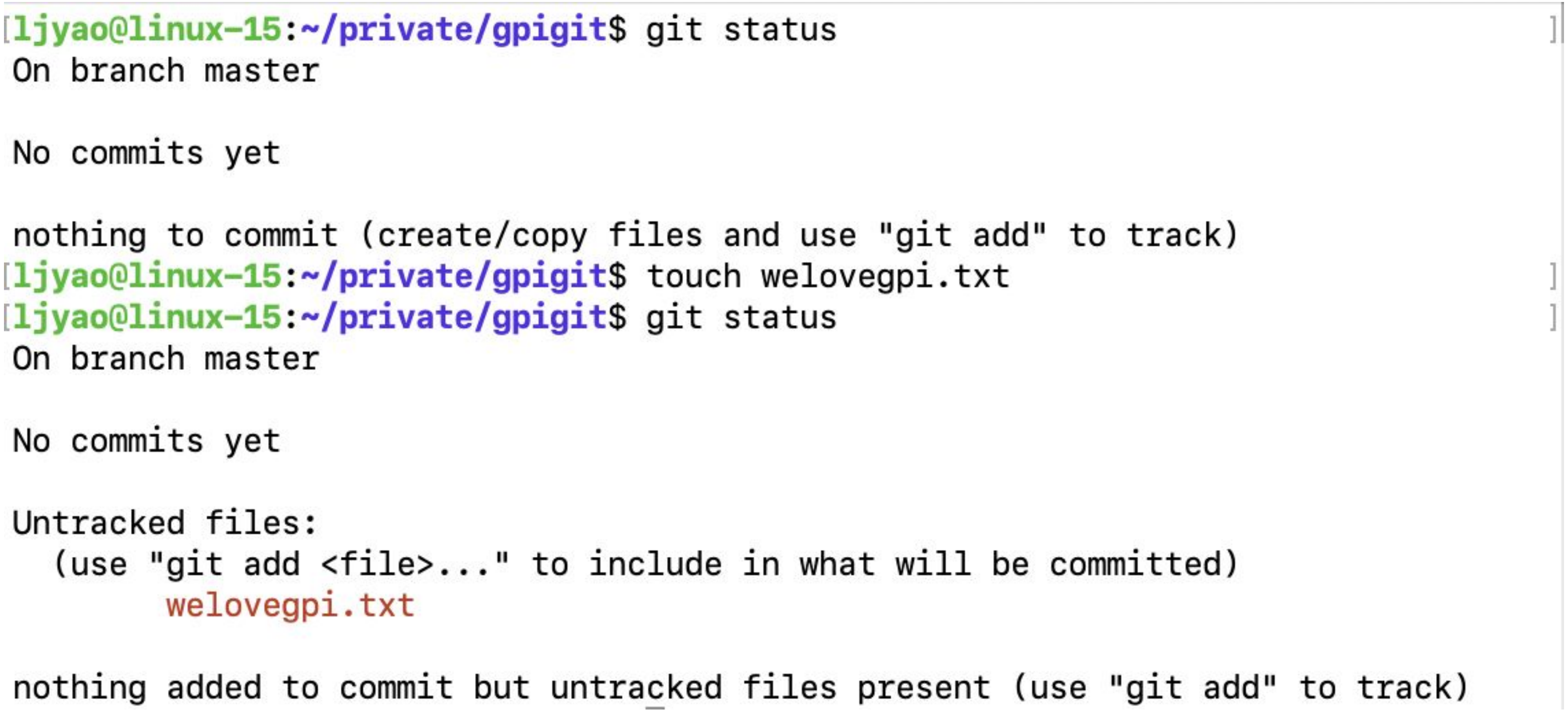
```
ljyao@linux-15:~/private/gpigit$ git init
Initialized empty Git repository in /afs/andrew.cmu.edu/usr10/ljyao/private/gpig
it/.git/
ljyao@linux-15:~/private/gpigit$ ls -a
.  ..  .git
```

Checking what git is doing?

- We can check what git is doing with our files 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

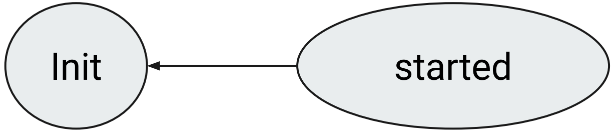
- Some files we want git to keep track of (e.g. our code)
- Some files we want git to ignore (log files, compiled files, etc.)
- git will NOT track anything unless you tell it to!
- To tell git to track a file:
  - $ **git add [path to file or folder]**
- If you want to add all changes/files in the current folder, we often use:
  - $ **git add .**
- What if you want to add all changes/files except one or two?
  - Add these files you want to ignore to the .gitignore file



```
[ljyao@linux-15:~/private/gpigit$ git status
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)
[ljyao@linux-15:~/private/gpigit$ touch welovegpi.txt
[ljyao@linux-15:~/private/gpigit$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        welovegpi.txt

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

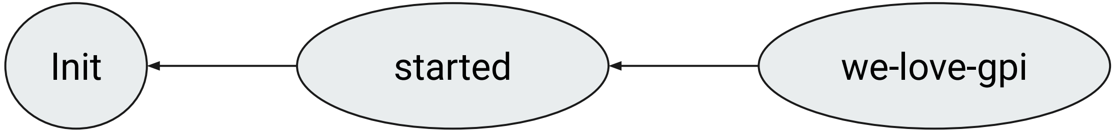
Commits: what are those?

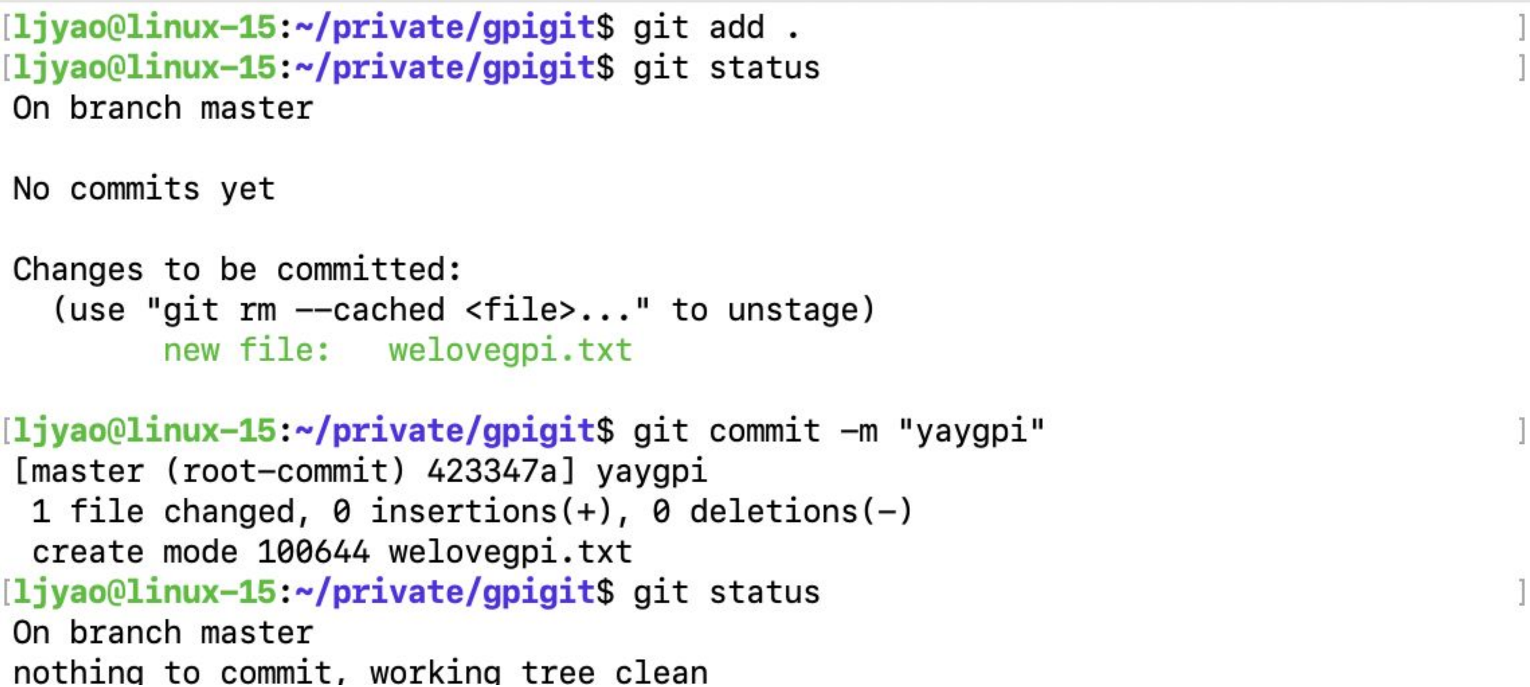
- Commits are a collection of changes that get added to the graph
- These are the snapshots that you are able to jump between
- You also get to write a message describing what the changes you made were
- You can commit by running:
    - $ **git commit**
        - Will open in some text editor (vim by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



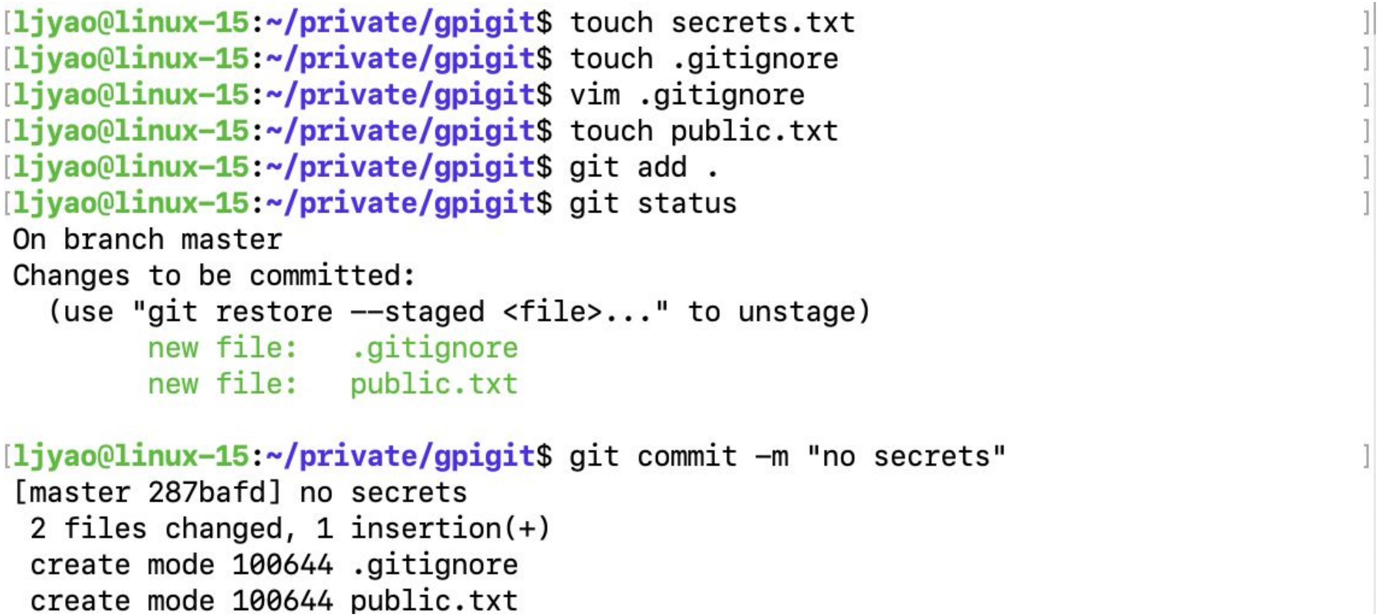
You can keep doing this as you make changes

- 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





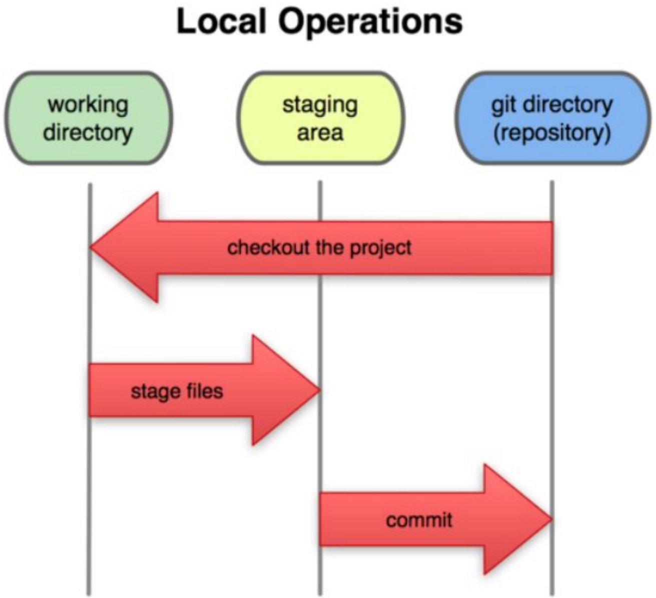
```
[ljyao@linux-15:~/private/gpigit$ git add .                                      ]
[ljyao@linux-15:~/private/gpigit$ git status                                     ]
 On branch master

 No commits yet

 Changes to be committed:
   (use "git rm --cached <file>..." to unstage)
         new file:   welovegpi.txt

[ljyao@linux-15:~/private/gpigit$ git commit -m "yaygpi"                         ]
 [master (root-commit) 423347a] yaygpi
  1 file changed, 0 insertions(+), 0 deletions(-)
  create mode 100644 welovegpi.txt
[ljyao@linux-15:~/private/gpigit$ git status                                     ]
 On branch master
 nothing to commit, working tree clean
```



```
[ljyao@linux-15:~/private/gpigit$ touch secrets.txt
[ljyao@linux-15:~/private/gpigit$ touch .gitignore
[ljyao@linux-15:~/private/gpigit$ vim .gitignore
[ljyao@linux-15:~/private/gpigit$ touch public.txt
[ljyao@linux-15:~/private/gpigit$ git add .
[ljyao@linux-15:~/private/gpigit$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   .gitignore
        new file:   public.txt

[ljyao@linux-15:~/private/gpigit$ git commit -m "no secrets"
[master 287bafd] no secrets
 2 files changed, 1 insertion(+)
 create mode 100644 .gitignore
 create mode 100644 public.txt
```

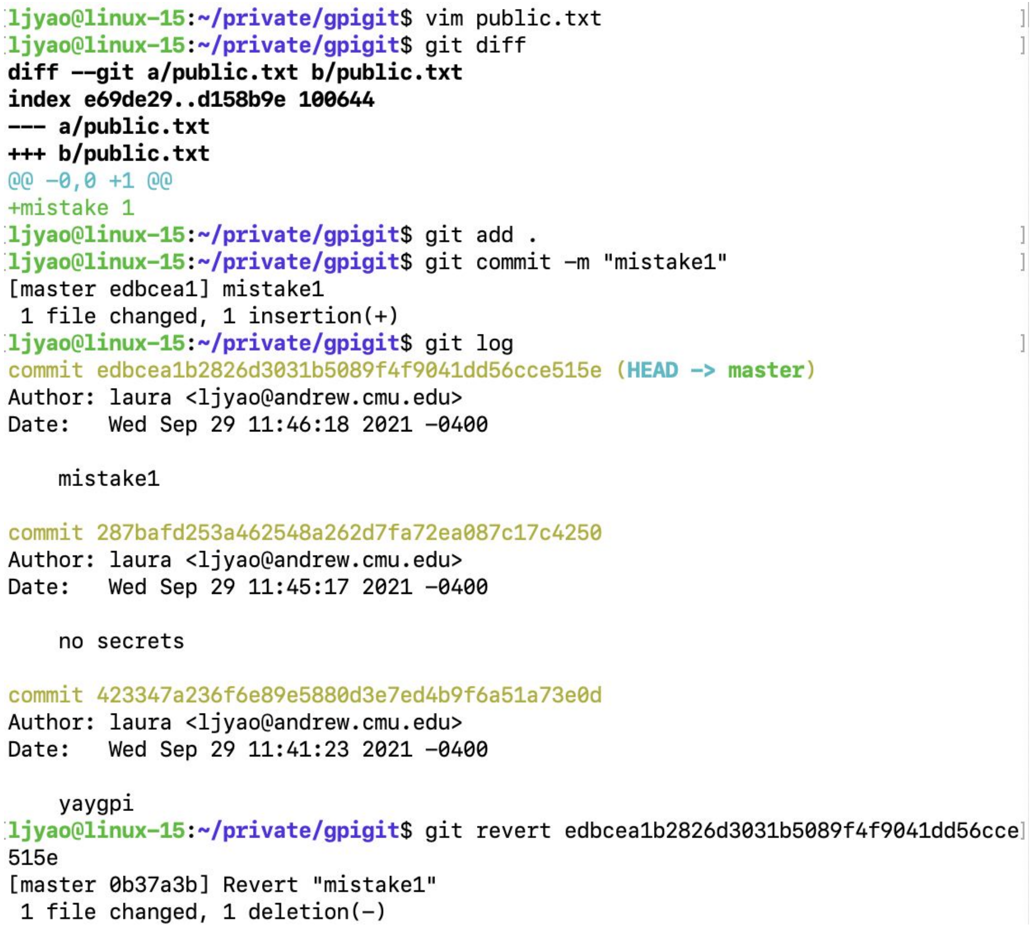
What's the process we just did?

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!!



Reverting

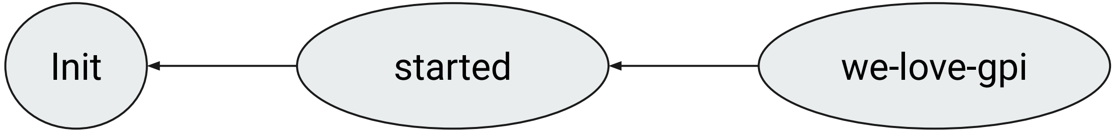
- $ **git log**
  - Checks your previous commits and lists them
- $ **git revert <commit-hash>** reverts a commit
  - by making a new commit with opposite changes
  - Doesn’t actually go backwards
- We will talk about git reset next week!



```
ljyao@linux-15:~/private/gpigit$ vim public.txt
ljyao@linux-15:~/private/gpigit$ git diff
diff --git a/public.txt b/public.txt
index e69de29..d158b9e 100644
--- a/public.txt
+++ b/public.txt
@@ -0,0 +1 @@
+mistake 1
ljyao@linux-15:~/private/gpigit$ git add .
ljyao@linux-15:~/private/gpigit$ git commit -m "mistake1"
[master edbcea1] mistake1
 1 file changed, 1 insertion(+)
ljyao@linux-15:~/private/gpigit$ git log
commit edbcea1b2826d3031b5089f4f9041dd56cce515e (HEAD -> master)
Author: laura <ljyao@andrew.cmu.edu>
Date:   Wed Sep 29 11:46:18 2021 -0400

    mistake1

commit 287bafd253a462548a262d7fa72ea087c17c4250
Author: laura <ljyao@andrew.cmu.edu>
Date:   Wed Sep 29 11:45:17 2021 -0400

    no secrets

commit 423347a236f6e89e5880d3e7ed4b9f6a51a73e0d
Author: laura <ljyao@andrew.cmu.edu>
Date:   Wed Sep 29 11:41:23 2021 -0400

    yaygpi
ljyao@linux-15:~/private/gpigit$ git revert edbcea1b2826d3031b5089f4f9041dd56cce
515e
[master 0b37a3b] Revert "mistake1"
 1 file changed, 1 deletion(-)
```

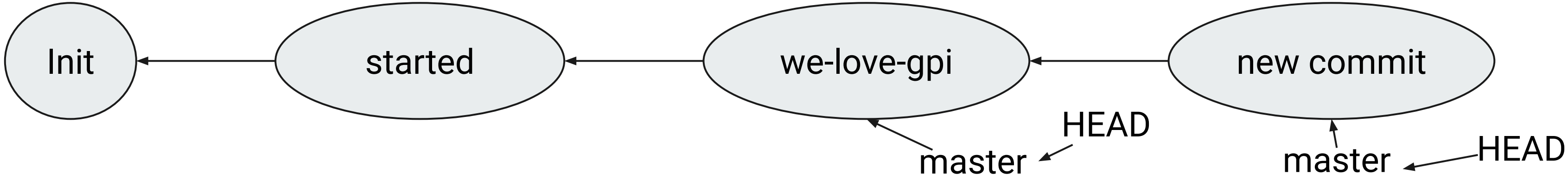
There is this tree thingy, how do i see it?

- $ **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 in a pretty format
- Do you notice the git hashes?
  - They look like this: 06a12a2465b78ca92f08aacf774cb98fda3c3519
  - They will be useful later



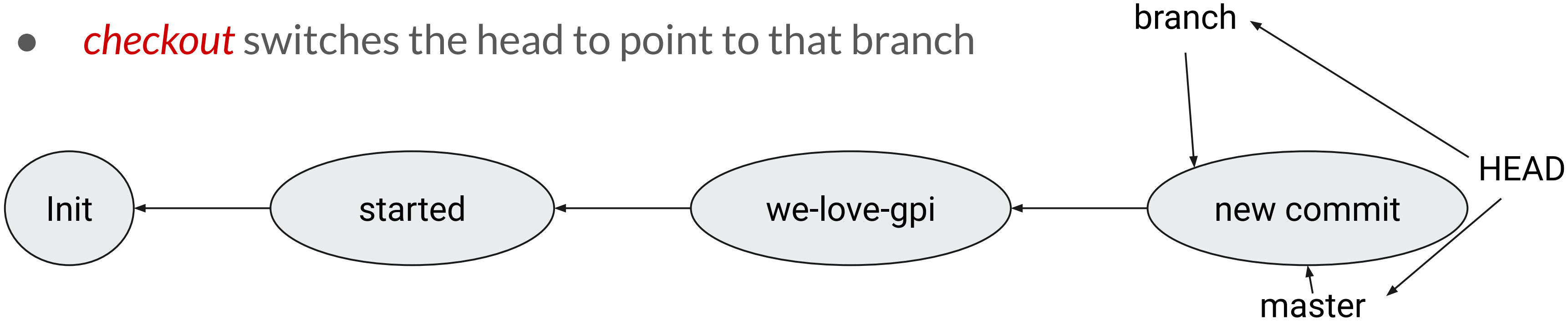
Aren't all these trees straight lines?

- 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



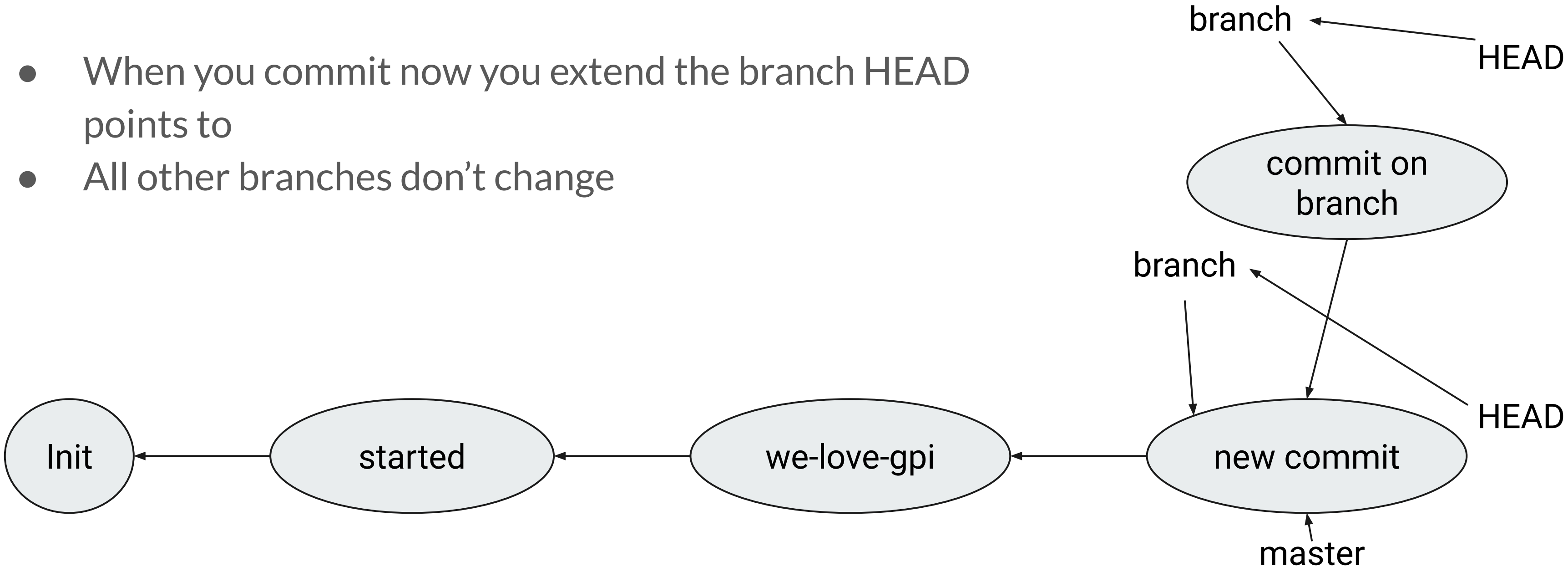
Making branches

- 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



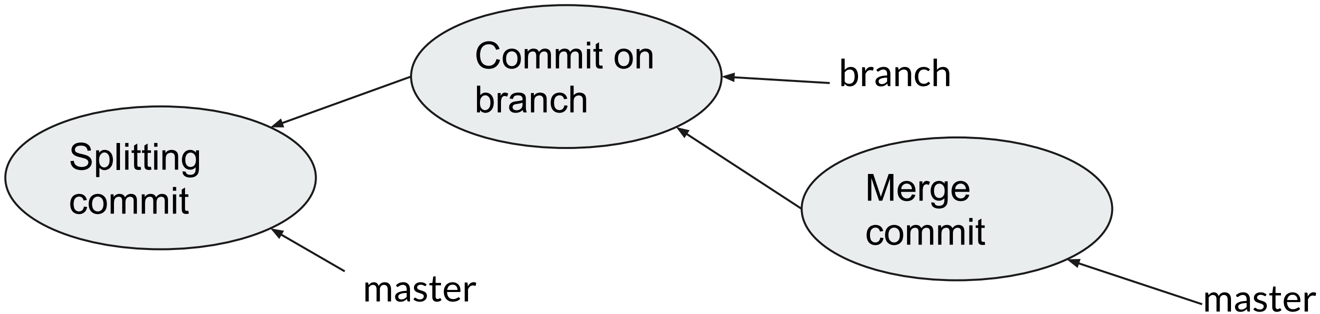
Making branches

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



Combining branches

- Branches let multiple people work on different parts of the project without breaking each other!
- When you want to combine two branches:
  - ○ $ **git merge [branch you want to merge]**
- This makes a commit that both branches and HEAD point to

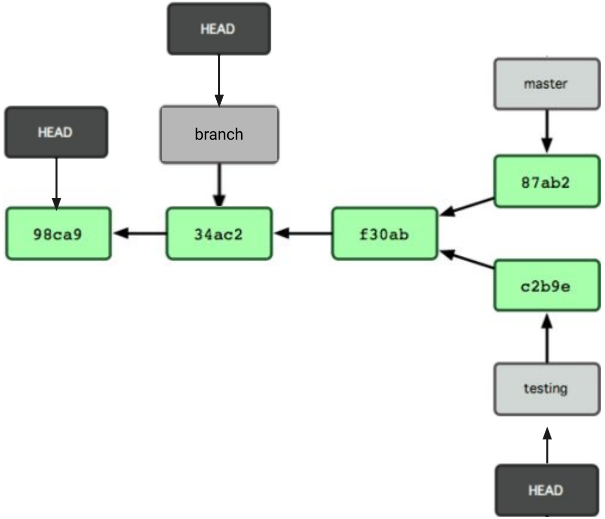


How to actually merge branches?

- 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

Reminders

- **WizardLab** due 11:59 pm ET tonight
- Extratation this Saturday 1-2 pm at GHC 4211: **Exam Review**
- Course Feedback on tinyurl.com/f21-gpi-feedback

Helpful hints for romancelab

- 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!
- 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!!!