

Git Basics

Tony Cui & Abby Chou

How do we collaborate on code?






Doe

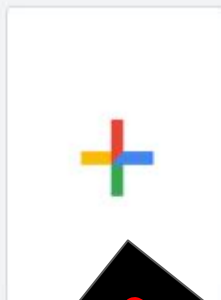


Search



Start a new document

Template gallery 



Blank q



Resume
Serif



Resume
Coral



Letter
Spearmint



Project proposal
Tropic



Recent

code

Owned by anyone ▼



Lecture Title: Version Control and Git

Prepared by: Tony and Alvin

Lecture Description & Summary

Git is a powerful version control tool for both collaboration, managing projects and tracking code changes. It's all you need to get started. Git is free and open source software. It's designed to work across multiple branches in a project. It will also differentiate local and remote git repositories, and how it results in collaboration.

Because it is the first lecture, we will also go over basic terminal usage, such as moving in and out of directories, creating new folders, and listing the current directory.

High-Level Learning Objective

- Only give vehicles numbering slots so we follow the convention of check-out (you'll see the check-out slot, not the check-in slot).
- Run the circuiting logic that arranges for expanding check-out.
- Queue at the end of the road.
- We figure out how to move the vehicle on the green and code on a red light.
- Maybe point out the point-to-point code to switch back and forth between the two.
- Add a sort-of bus model between regular.
- Expand the Route section a bit (possibly 2).
- Maybe clarify what happens if you don't have your oranges and then get worried/lost.
- Maybe add a comment line in the middle.

- *426. nefe* *homage*
 n. → it is possible to say the image is like (a good)

- Mrs. V reported that head is dull (3/10), steady but not actually increased w/

Backwards Design Worksheet

© 2005 Blackwell Publishing Ltd, *Journal of Internal Medicine* 258: 103–110

Q4 What would students learn better & more into your subject

- Items I read apps are structured with both forward and backward (i.e. what logic goes on the forward and backward, and more specifically, which logic goes into which block)
- Necessary to understand the module: and quick so we can review items jointly that include
 - How to write, and evaluate with you and me
 - Logic flow
 - Conditional handling
 - Common functions

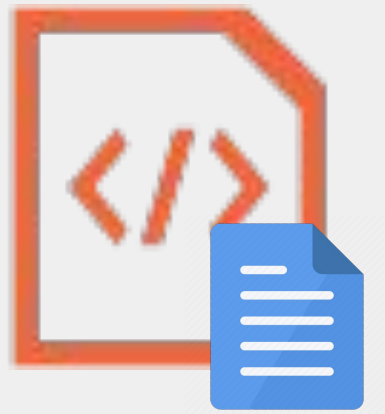
(c) When a student is not present, a teacher should give that work back.

- flow username/password authentication across in a high level
- Google sign-in with fine print details
 - telling ahead of time that when the client for who will be authenticating

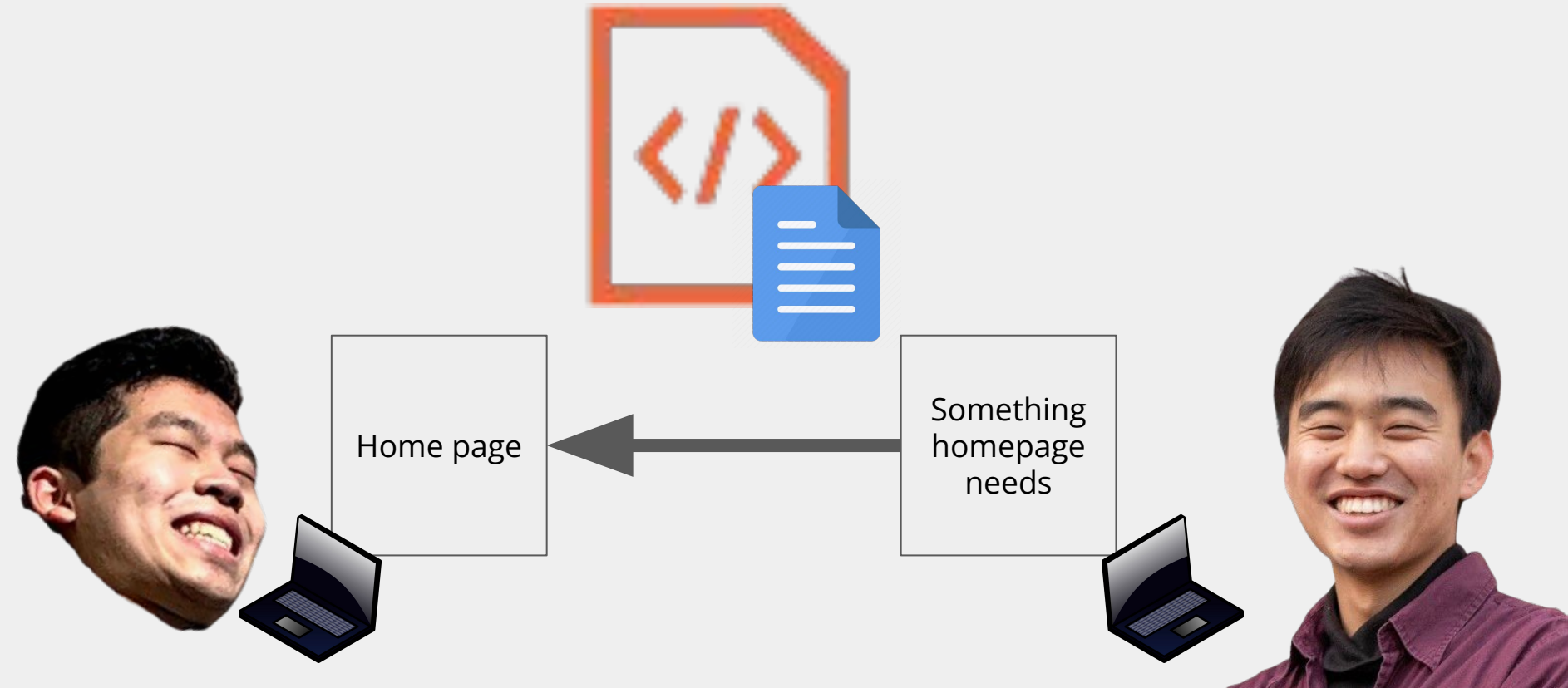
Consider
the
following



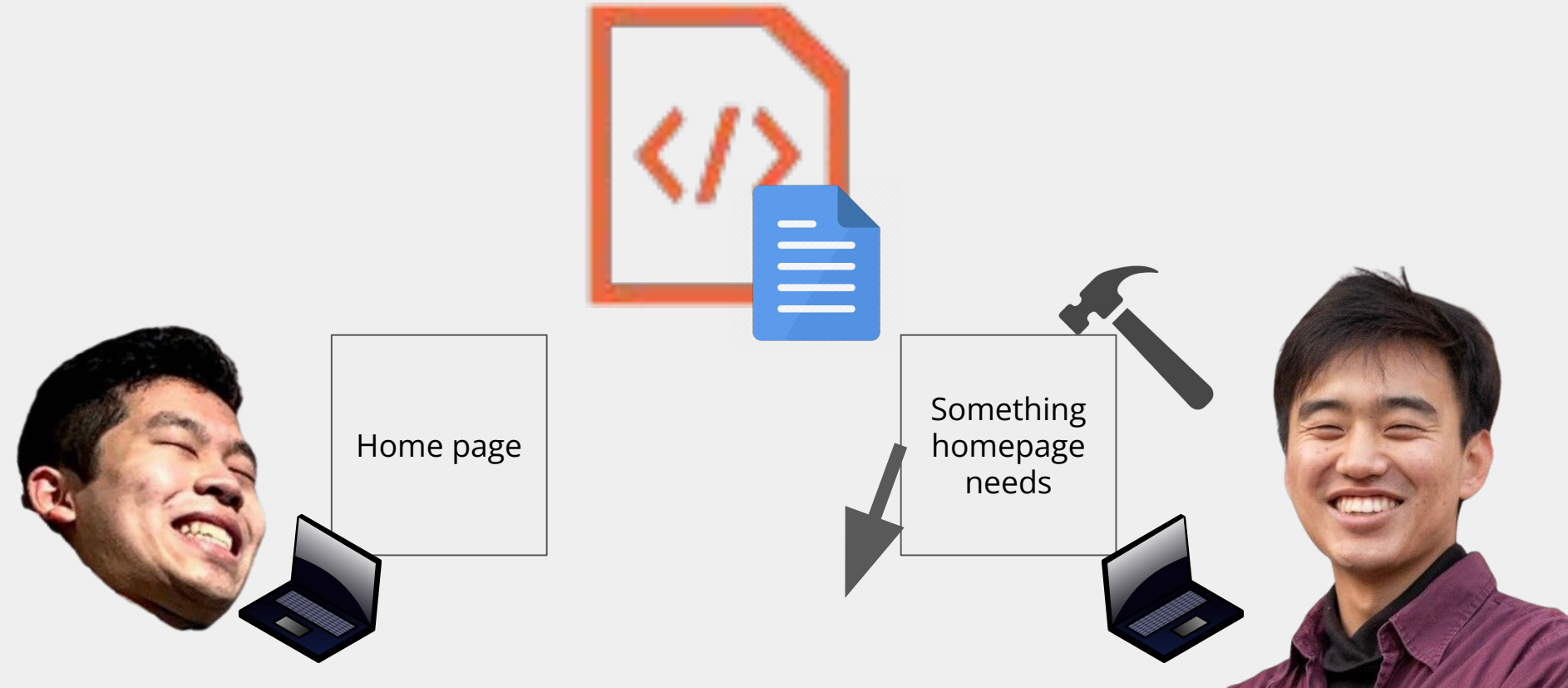
The issue with Code-Sharing



The issue with Code-Sharing



The issue with Code-Sharing

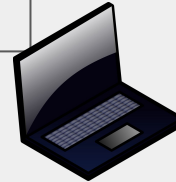


The issue with Code-Sharing

Bro what happened?

Home page

Something homepage needs



The issue with Code-Sharing

Bro what happened?

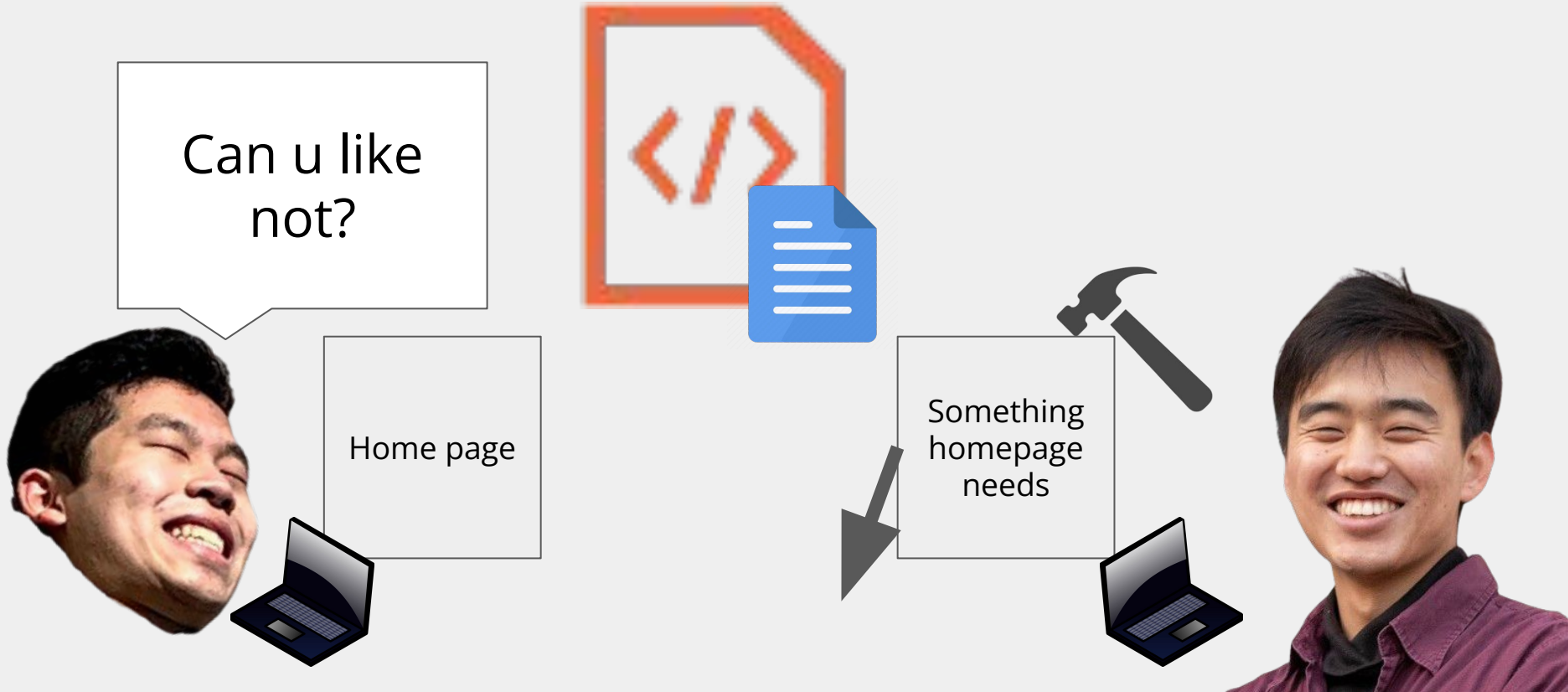
Home page



✨ upgrading ✨

Something homepage needs

The issue with Code-Sharing



The issue with Code-Sharing

Can u like
not?

Home page



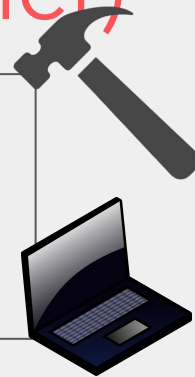
Can u like
deal with it

Something
homepage
needs

We need independent copies of the code! (so people can work without disrupting each other)

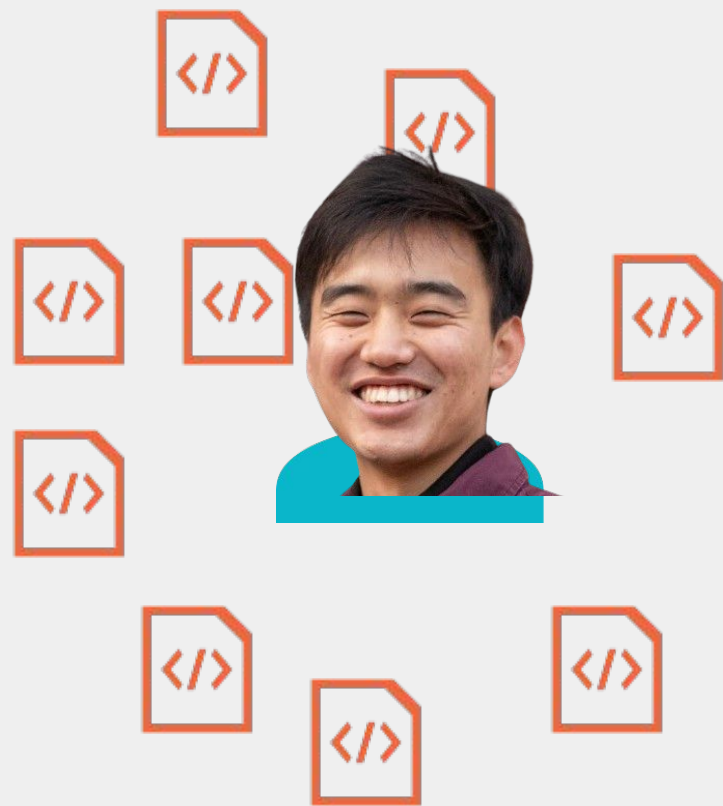
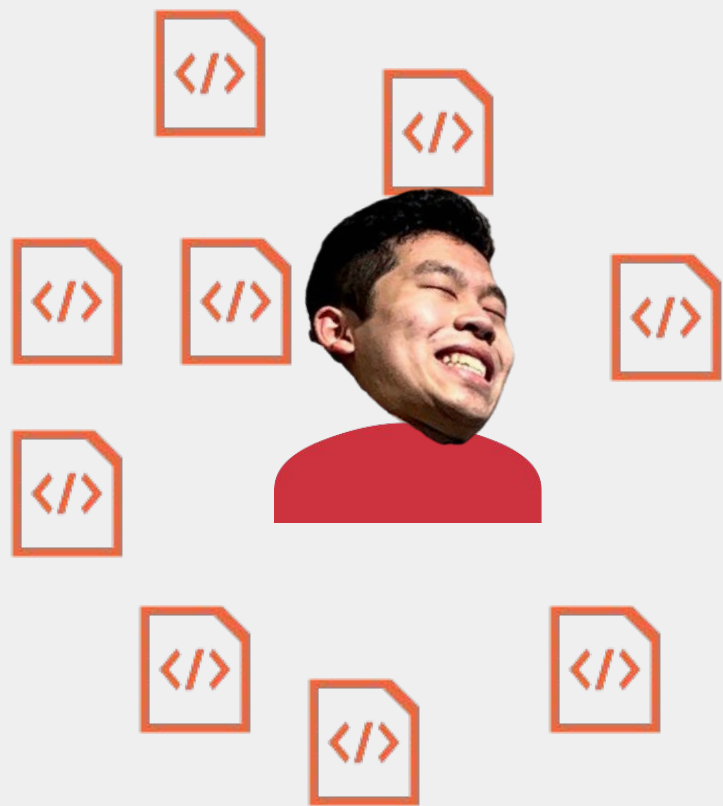


Who
broke xyz
i need it
rn >:(



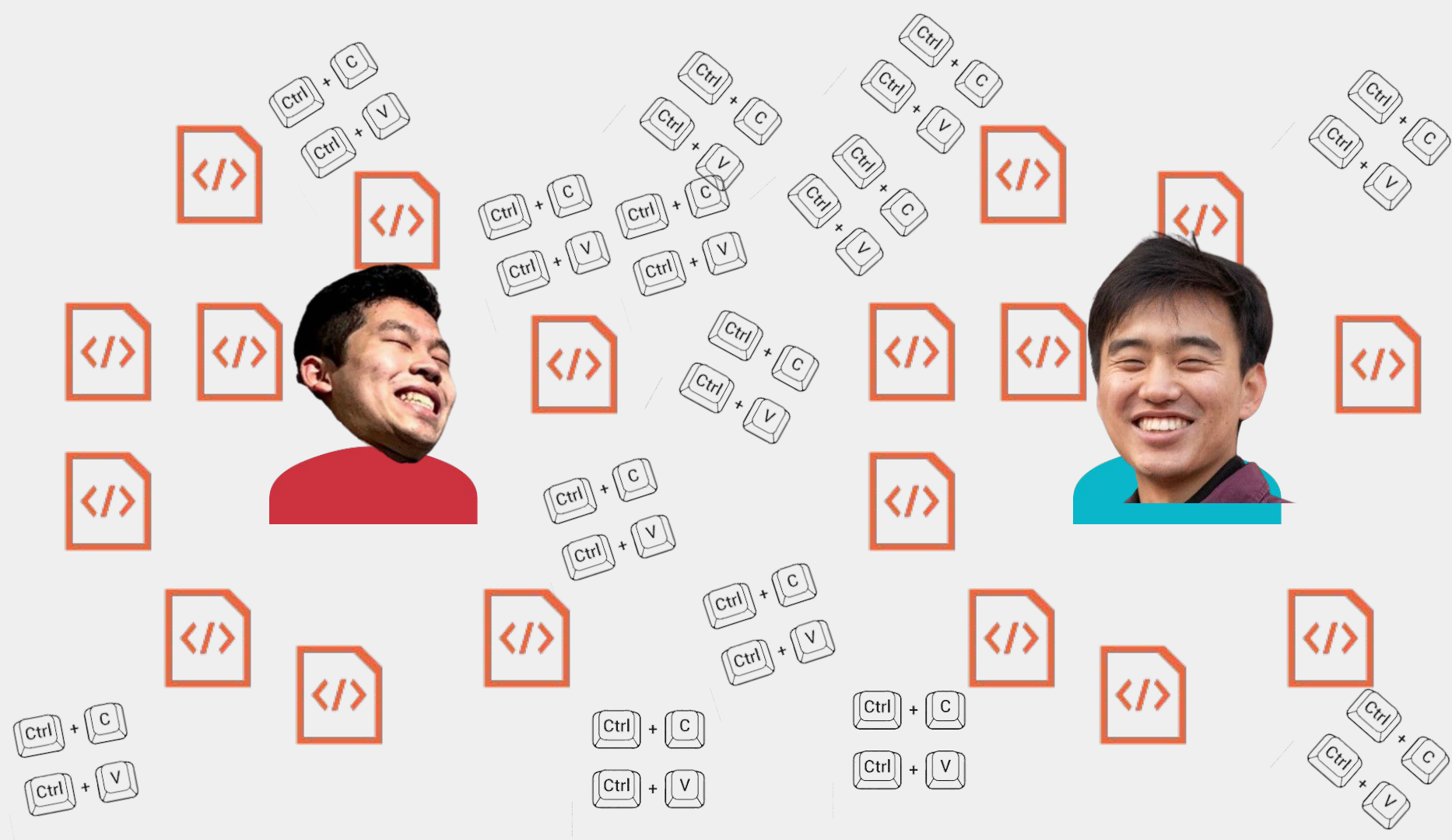
Something
homepage
needs

...but that creates so many problems 🤔

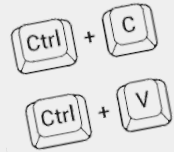
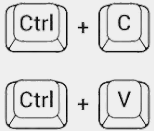
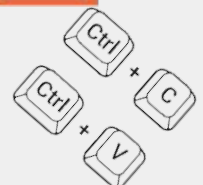
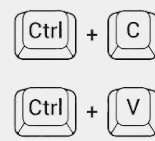
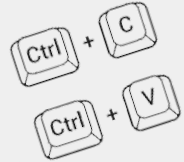
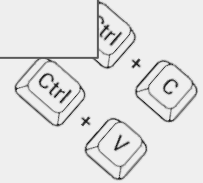
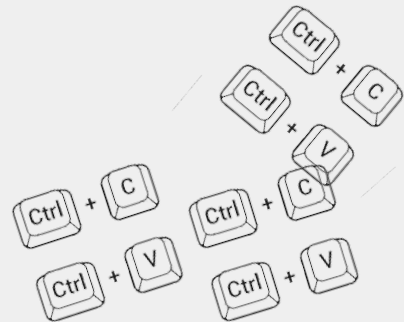
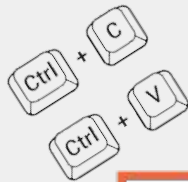
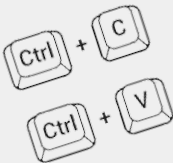


Hey I know you made changes in like 50 places
in that file but can you copy the changes over
one by one





Remove lines 250-280
Add 24 new lines



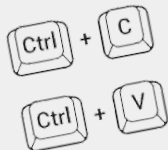
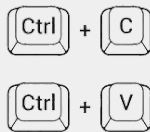
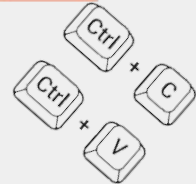
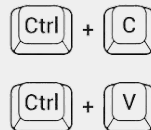
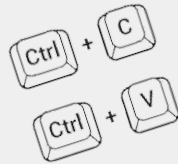
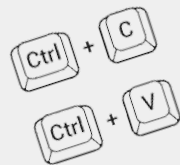
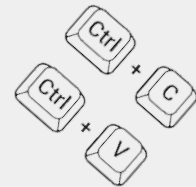
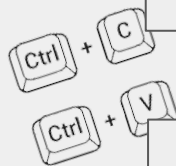
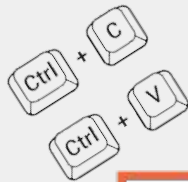
An hour later...

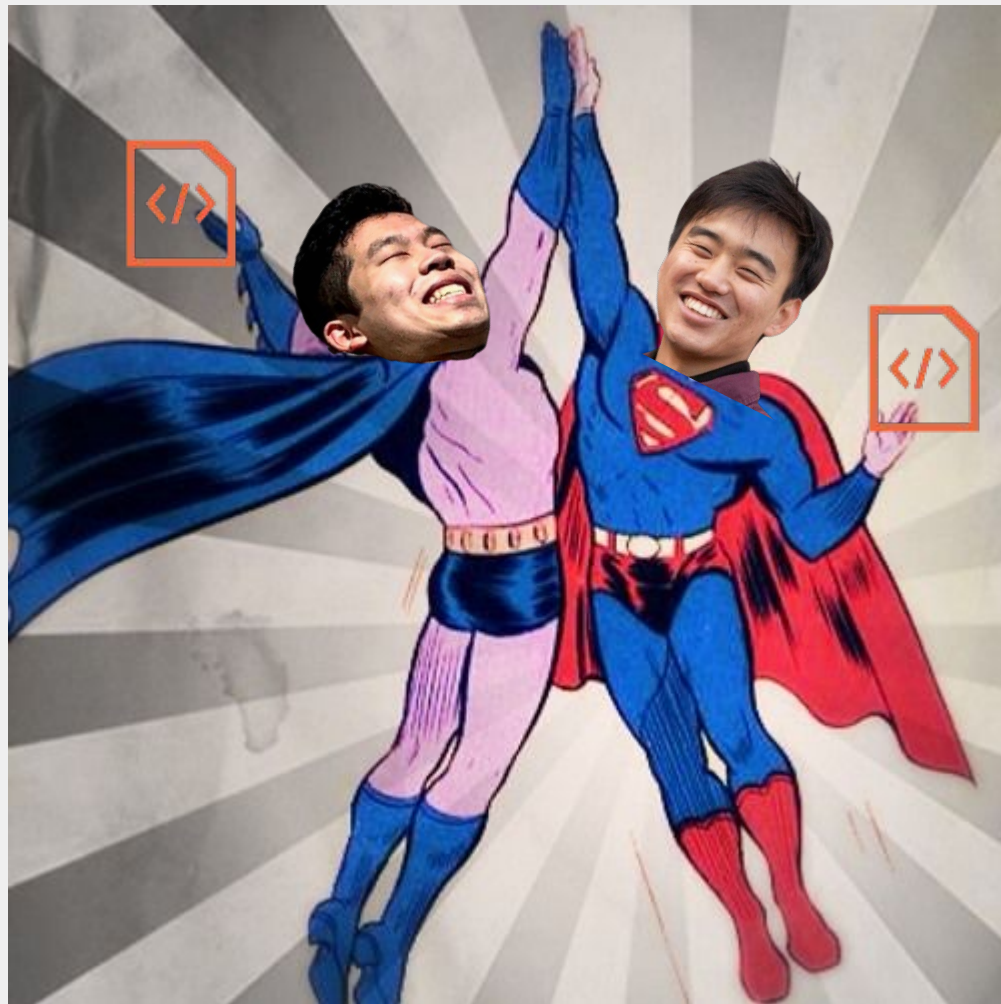
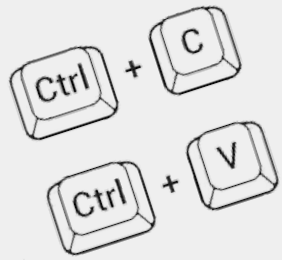
WAIT!!!!!!!!!!
I NEEDED
LINES 250-280

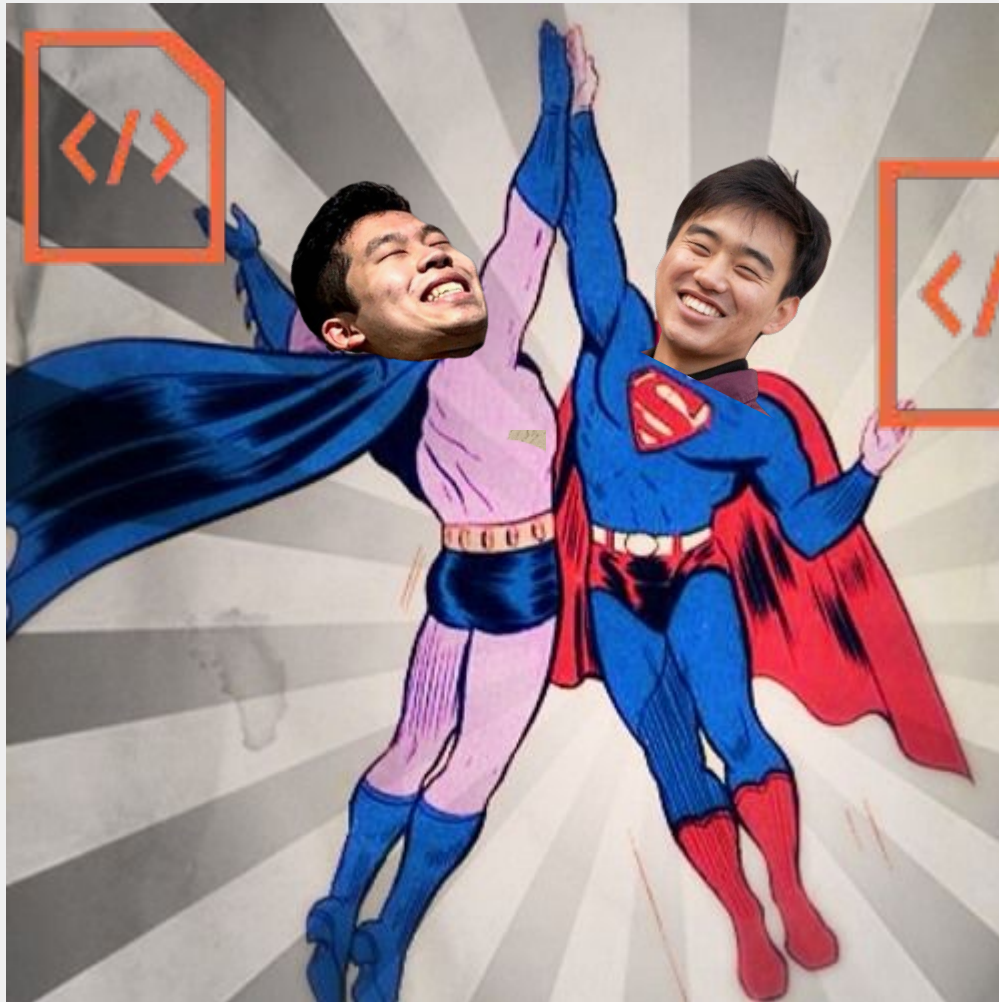
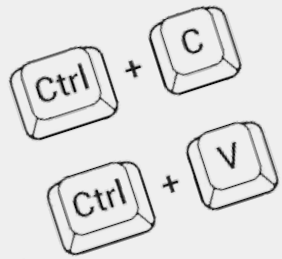


ROLLBACK ROLLBACK

WHERE WHERE







????



The Problems

- Need independent local copies of the codebase
- Need to be able to merge different people's changes together
- Need to keep track of versions
- Need to know which version is the most up-to-date

How do we solve these issues?
Introducing...



git

What does Git do?

Git tracks *changes*.
(How? ✨math✨)

mydoc.txt

I'm writing a story.
It's really good!
And about cats!

version1



I'm writing a story.
It's really good!
And about cats!

Like corgis :D

version2

mydoc.txt

I'm writing a story.
It's really good!
And about cats!

version
1



I'm writing a story.
It's really good!
And about cats!
Like corgis :D

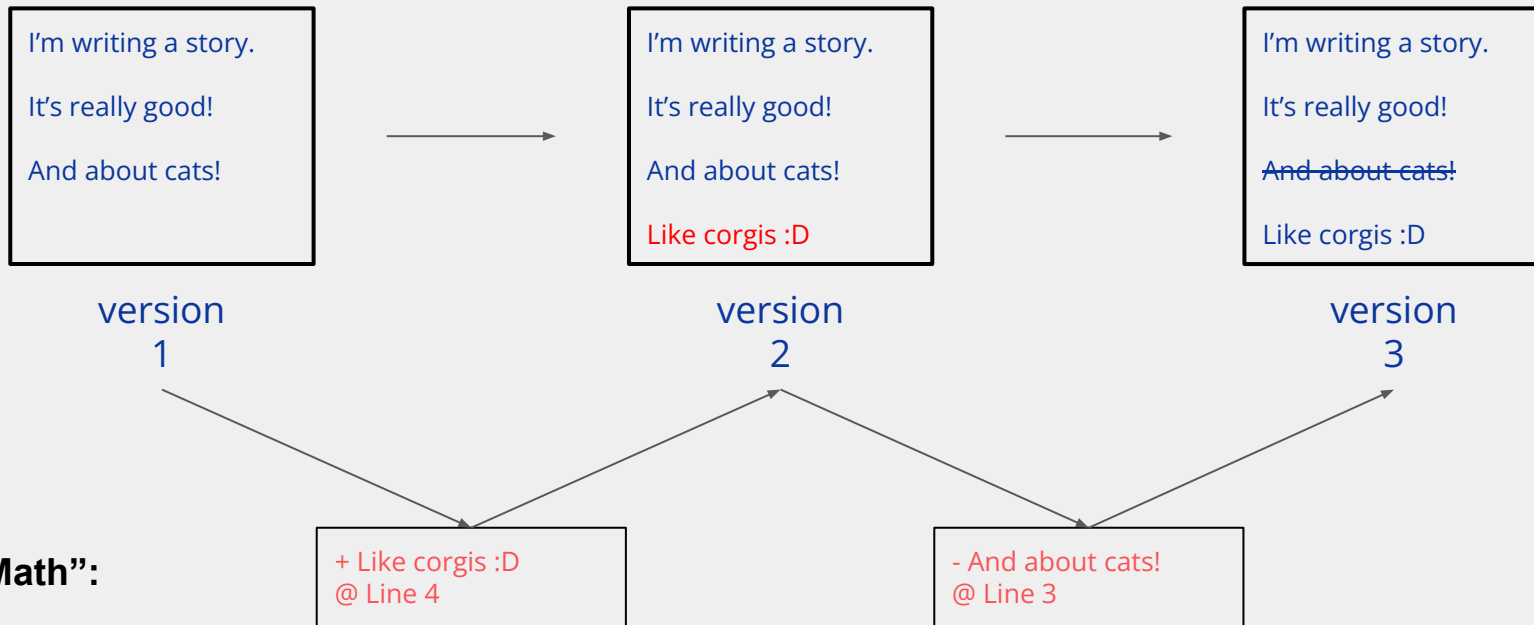
version
2



I'm writing a story.
It's really good!
~~And about cats!~~
Like corgis :D

version
3

mydoc.txt



The “Math”:

example

Getting set up with git :D

Demonstration

Try the following:

1. Make a new directory
2. Cd into that directory
3. Turn your directory into a git repo!

Reference

cd [directory] change directory

ls list the contents of current directory

mkdir [folder_name] make directory

git init turns current directory into git repo!

Special Characters:

. current directory

.. parent directory

Your turn!

Try the following:

1. Make a new directory
2. Cd into that directory
3. Turn your directory into a git repo!

Reference

cd [directory] change directory

ls list the contents of current directory

mkdir [folder_name] make directory

git init turns current directory into git repo!

Special Characters:

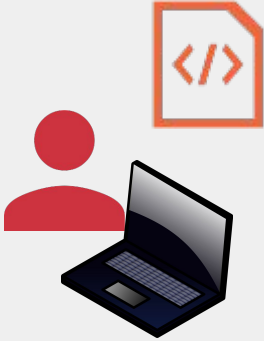
. current directory

.. parent directory

Staging & Committing

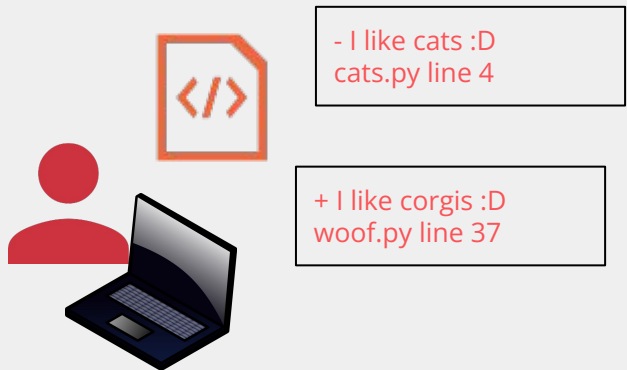


Staging & Committing



Working copy

Staging & Committing



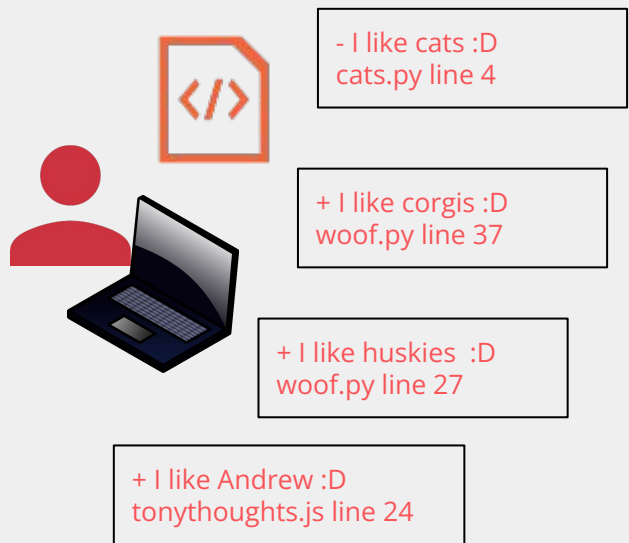
Staging & Committing



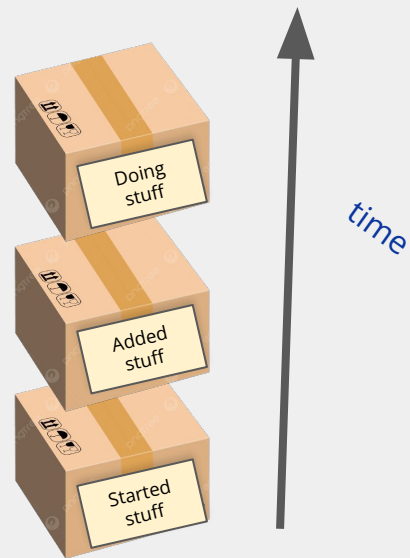
Staging & Committing



Staging & Committing



Working copy



Commit History

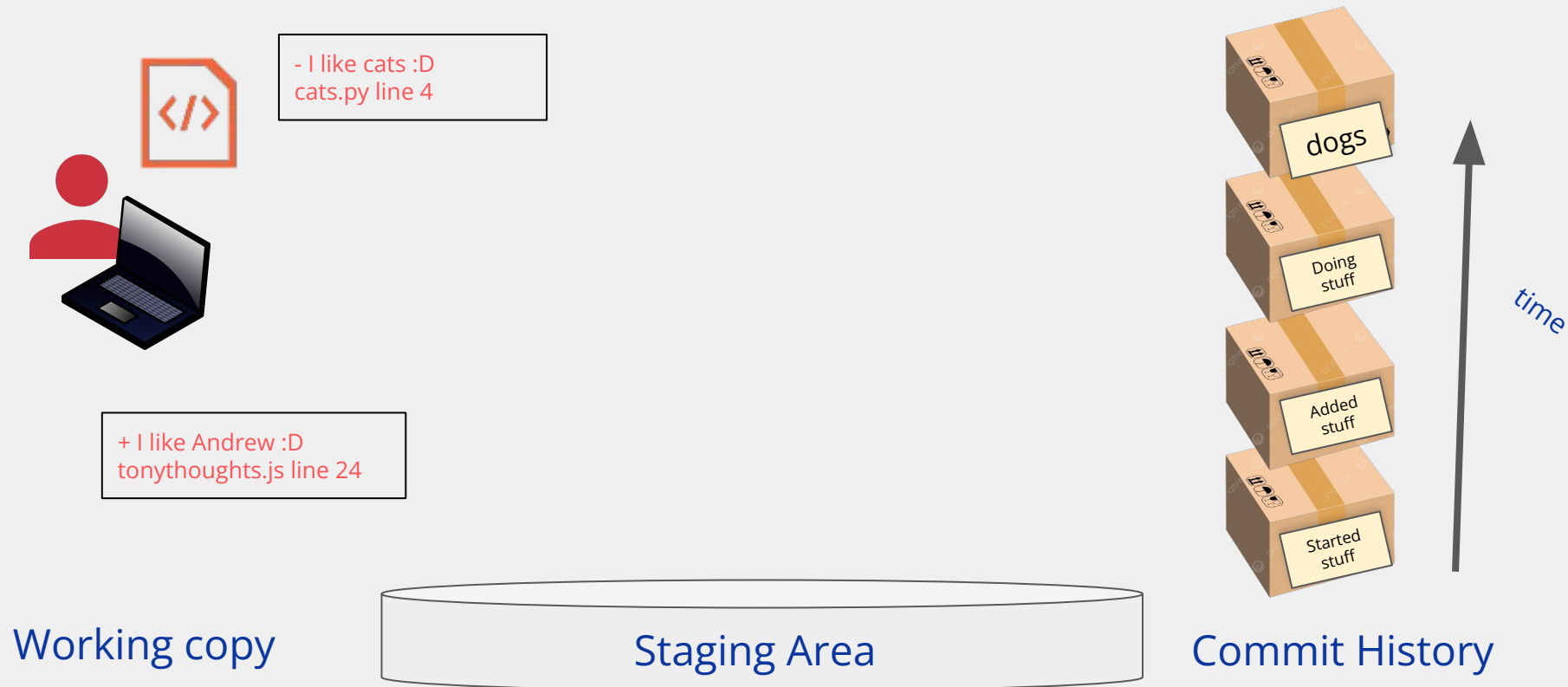
Staging & Committing



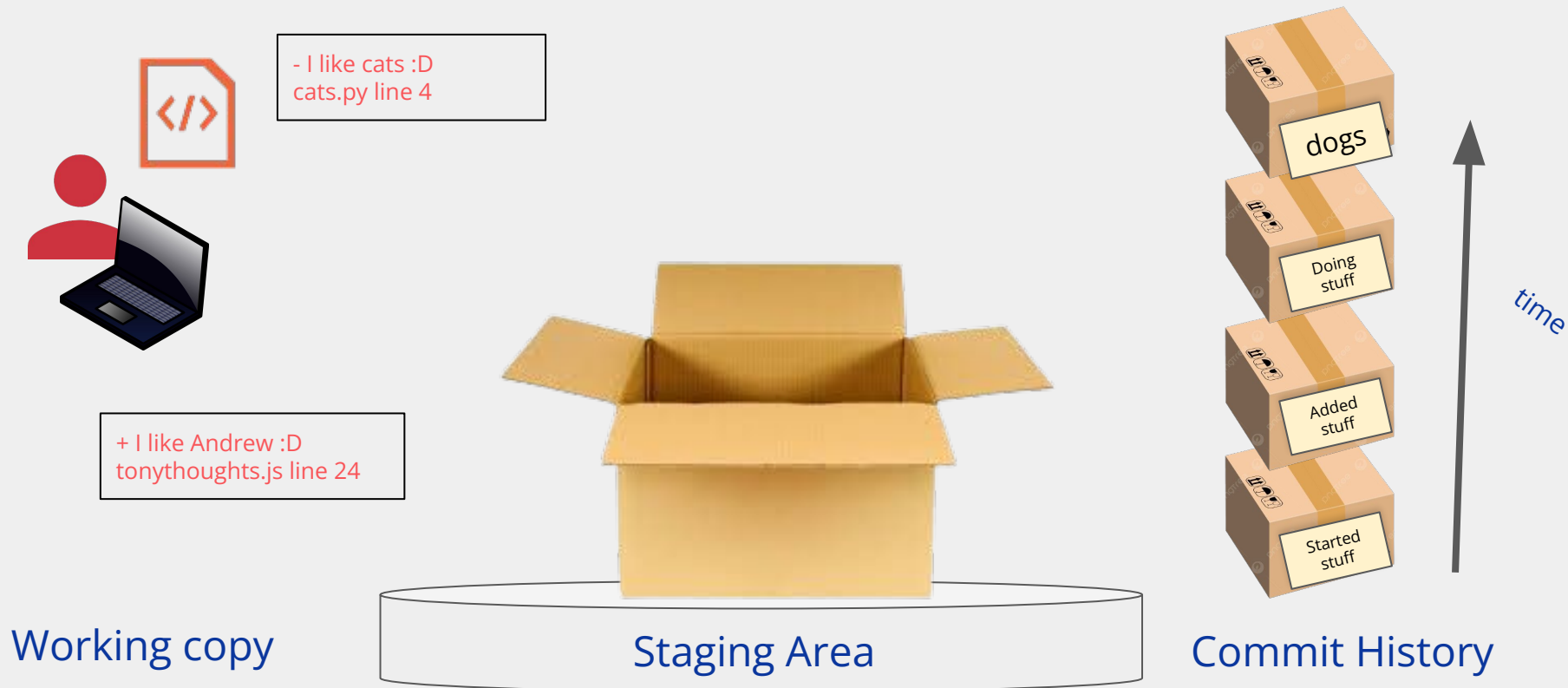
Staging & Committing



Staging & Committing



Staging & Committing



Demo Staging & Commits

Demonstration

1. Make some changes
 - a. Git status
2. Stage some changes
 - a. Git status
3. Make a commit
 - a. Git status
4. Git log
 - a. Git status

`git diff`

`git status`

`git add`

`git commit -m`

Your turn!

In your new repo...

1. Make some code changes
2. Stage changes with `git add`
3. Commit the changes
4. Check your commit log
5. (Optional) Repeat! (Try deleting stuff too)
 - a. After you make changes (before you stage them), `git diff` to see your changes!

Commands Ref

git add stage changes for commit

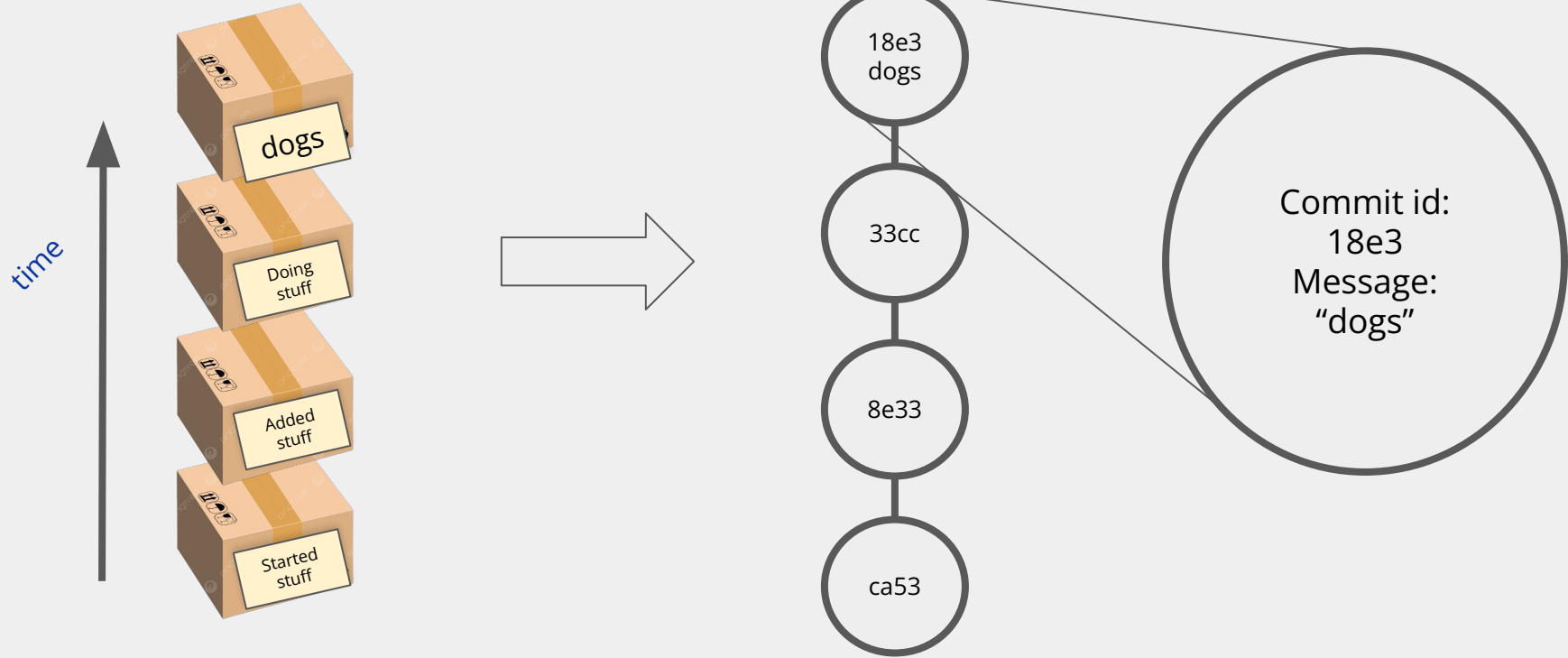
git commit -m "[msg]"

git status <- use this often!

git log show commit history

git diff show the diff (changes) between working copy and staged/committed copy

Object Graph

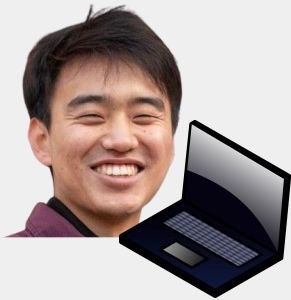


Consider
the
following



Uh oh, my boss is madge

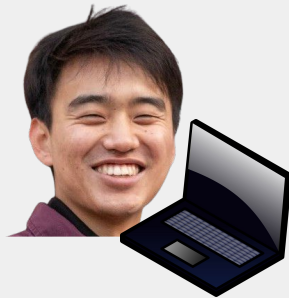
tony: "vibing working on my new and improved home page "



Uh oh, my boss is madge

tony: "vibing working on my new and improved home page "

andrew: "can u help me fix this bug on the home page uwu"

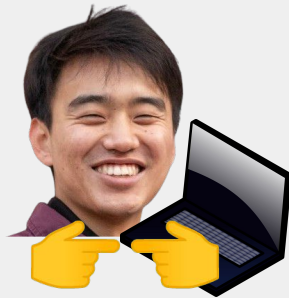


Uh oh, my boss is madge

tony: "vibing working on my new and improved home page "

andrew: "can u help me fix this bug on the home page uwu"

tony: "anything for u bb 🥰"



Uh oh, my boss is madge

tony: "vibing working on my new and improved home page"

andrew: "can u help me fix this bug on the home page"

tony: "anything for u bb 🥰"

boss abby (who got dumped recently 😞) : "tony we need a weblab-tinder feature TOMORROW and u hav to do it"



Uh oh, my boss is madge

tony: "vibing working on my new and improved home page"

andrew: "can u help me fix this bug on the home page"

tony: "anything for u bb 🥰"

boss abby (who got dumped recently 😞) : "tony we need a weblab-tinder feature TOMORROW and u hav to do it"



Uh oh, my boss is madge

tony: "vibing working on my new and improved home page "

andrew: "can u help me fix this bug on the home page"

tony: "anything for u bb 🥰"

boss abby (who got dumped recently 😞) : "tony we need a weblab-tinder feature TOMORROW and u hav to do it"

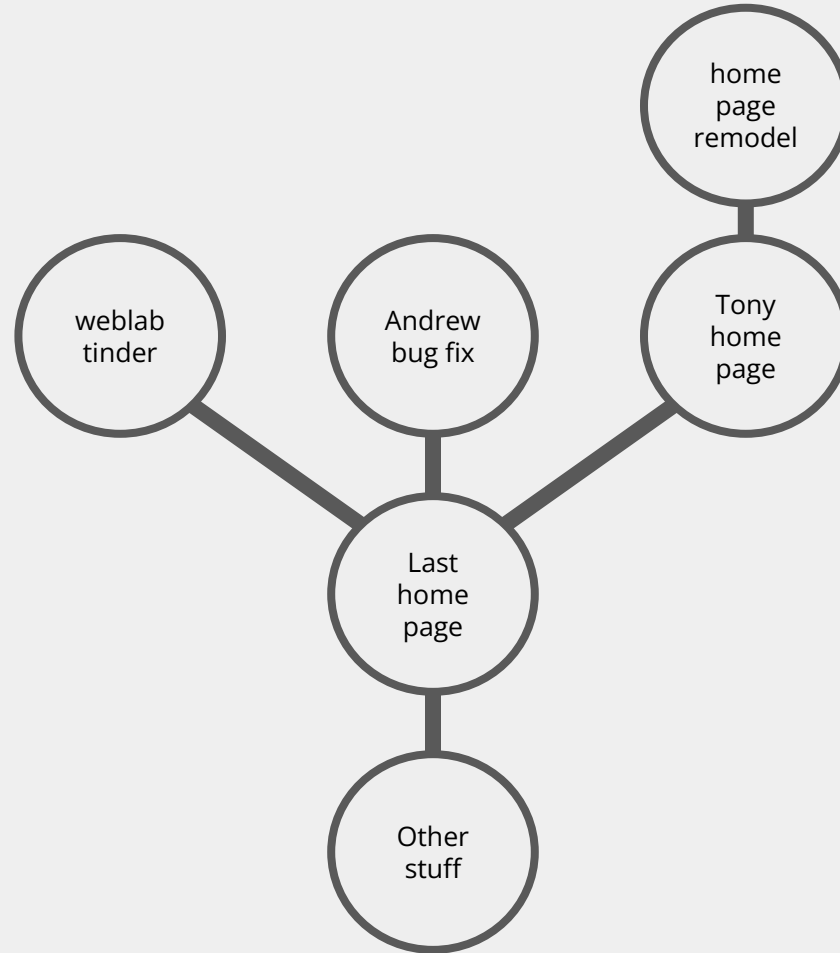
tony: "bruh"

How do we work on multiple features at once?

save me 😞



Branching



Demo Branches

Demonstration

1. Creating a new branch
2. Switching between branches

Your turn!

In your new directory...

1. Create a new branch
2. Add a couple of commits to your branch, check the log
3. Checkout back to main, check the log
4. Add a couple of commits to main, check the log
5. Try and draw and/or explain to your partner what you think the object graph looks like!

Handy commands

git branch see branches

git checkout [branch-name]

switch to existing branch

git checkout -b [branch-name]

create and checkout new branch

Previous Commands

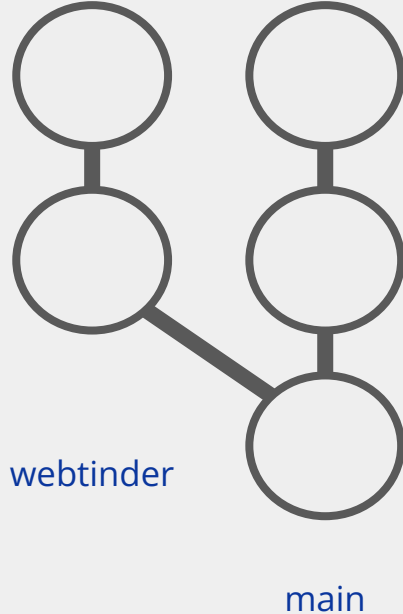
git status

git add

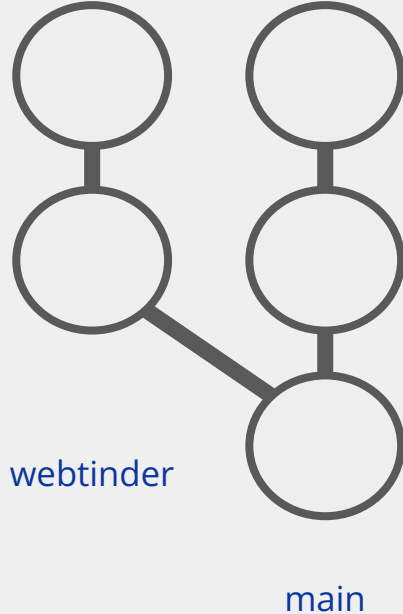
git commit -m

git log

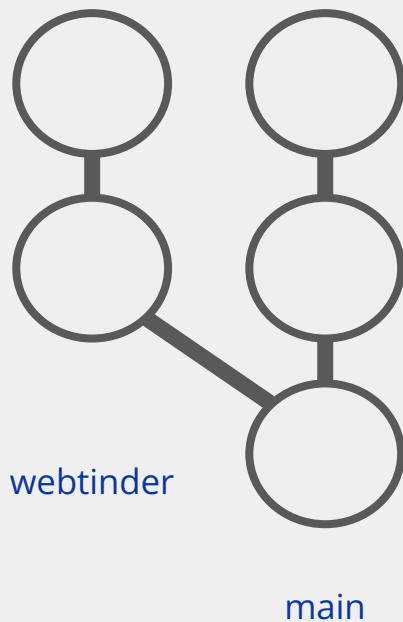
What is happening in the object graph :0?



What is happening in the object graph :0?



How do we put them back together?



example

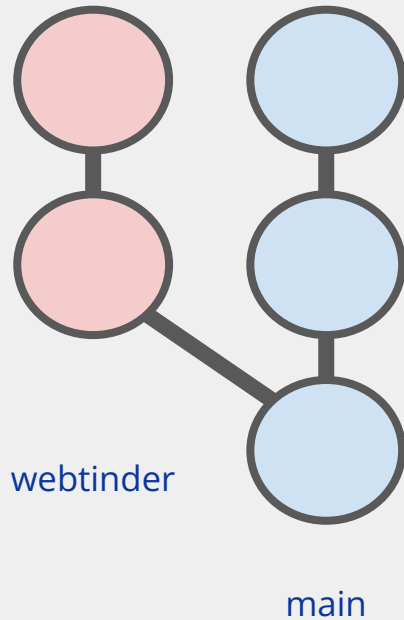
merging

Follow Along

Follow along!

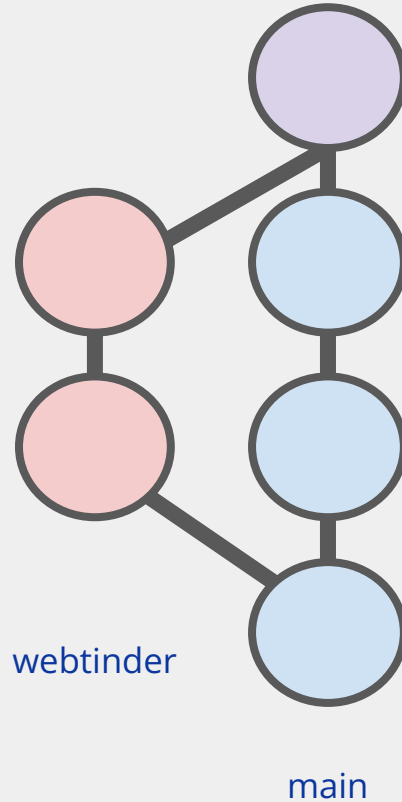
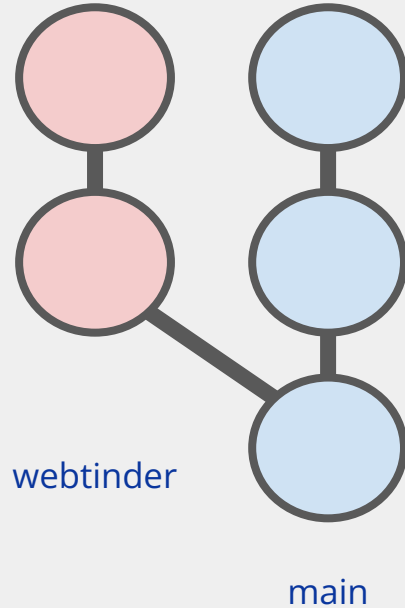
1. Checkout into main branch
2. Git merge [new branch name]

How do we put them back together?



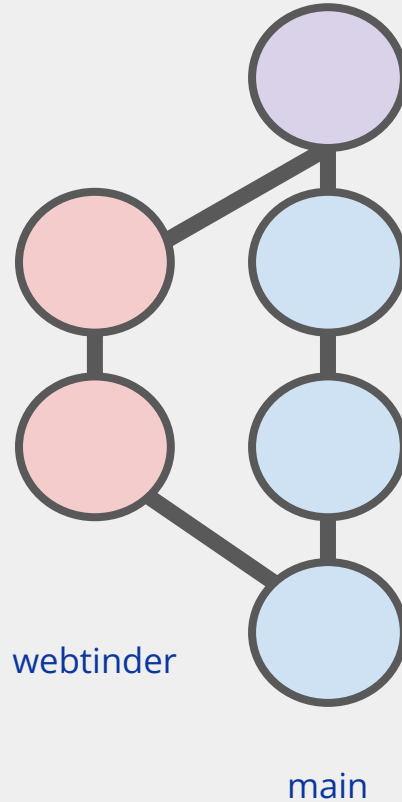
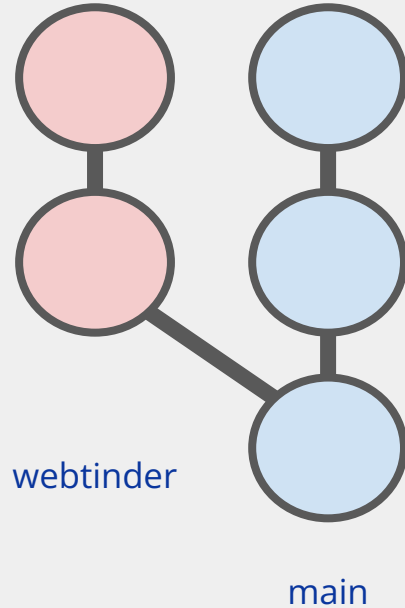
How do we put them back together?

sort of..



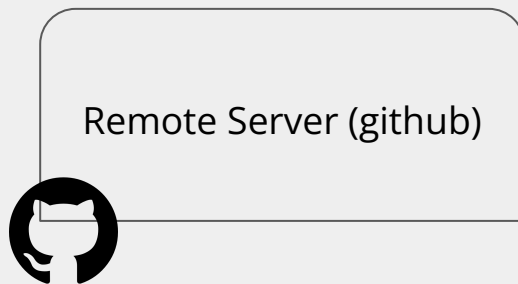
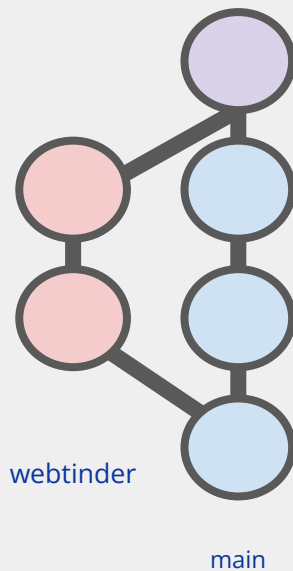
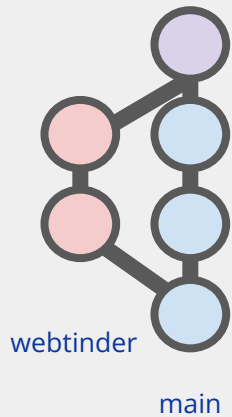
How do we put them back together?

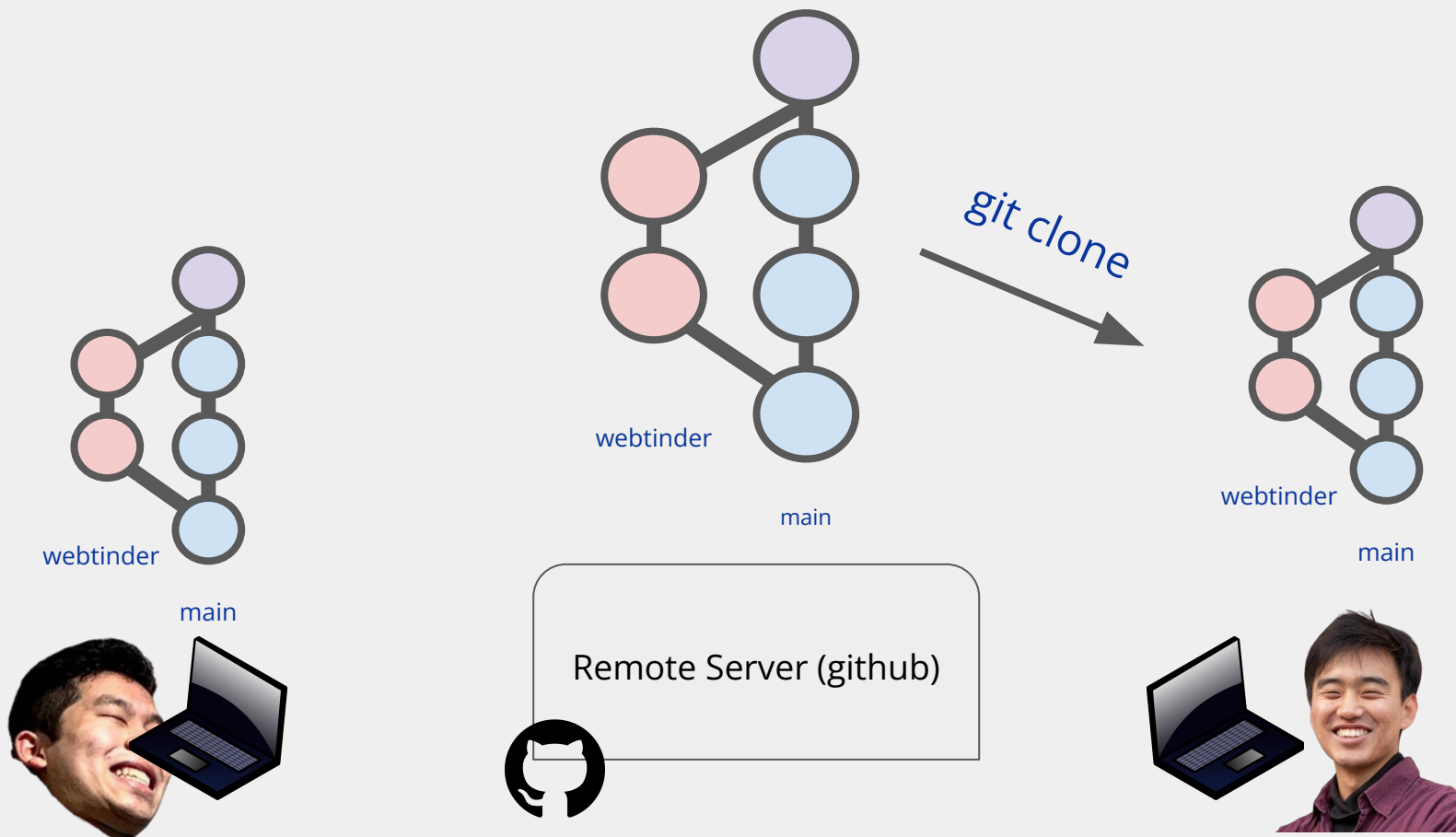
sort of..



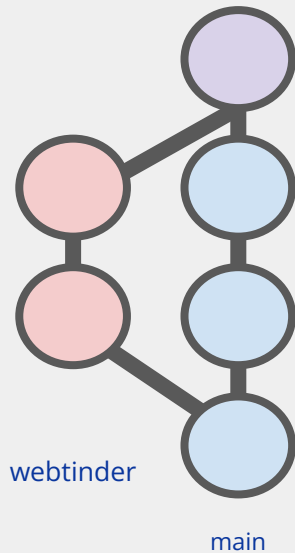
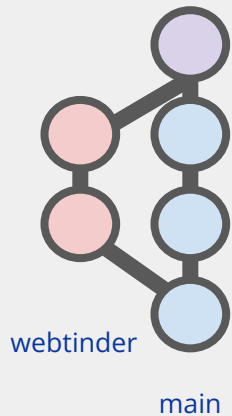
Preview

, how does collaboration
work then?

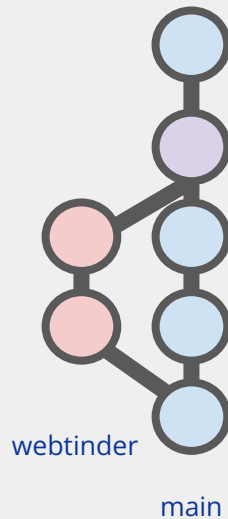


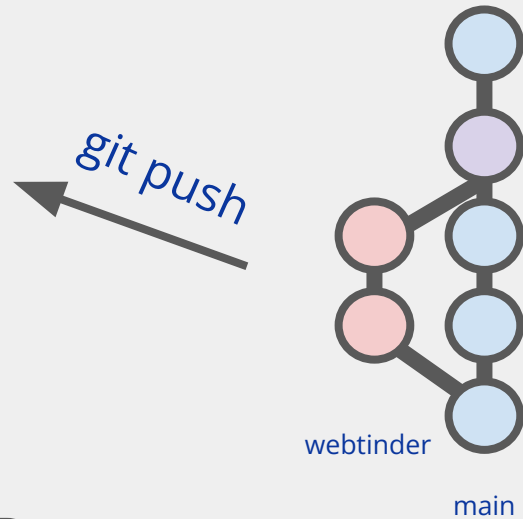
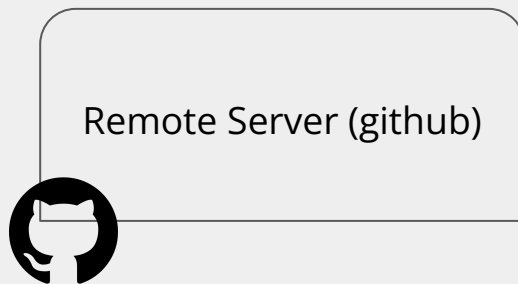
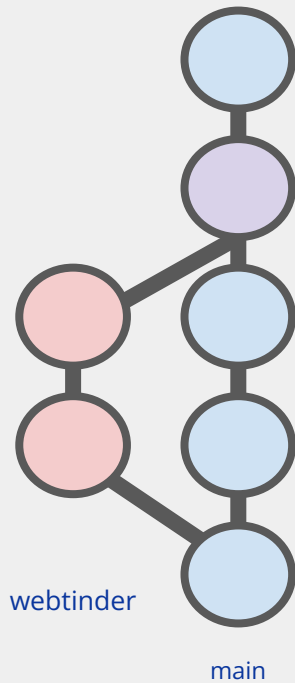
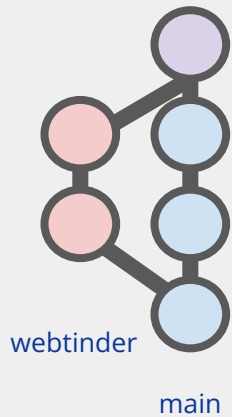


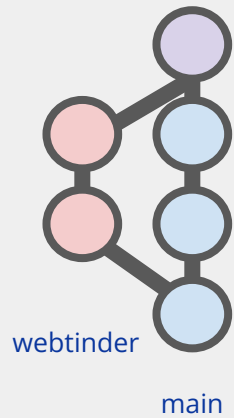
git commit



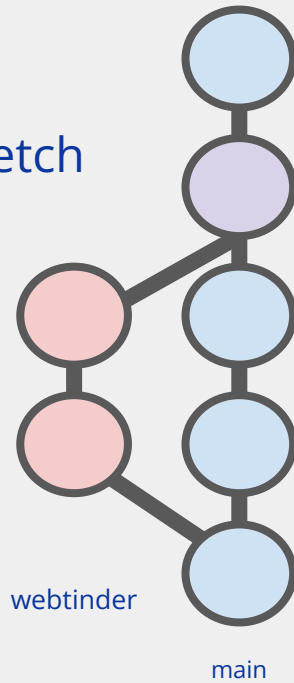
Remote Server (github)



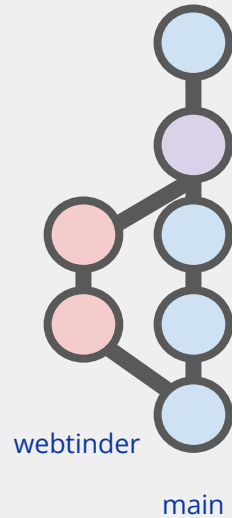


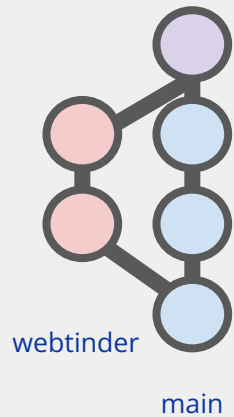


Git fetch

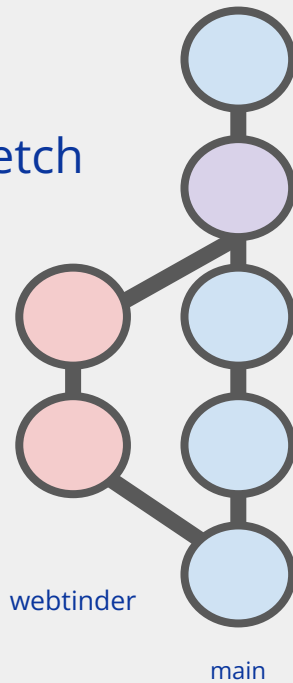


Remote Server (github)

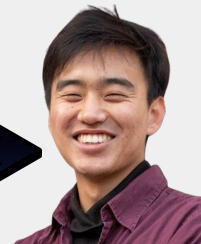
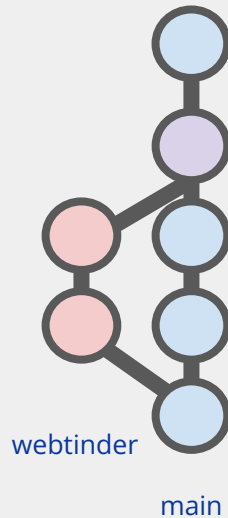


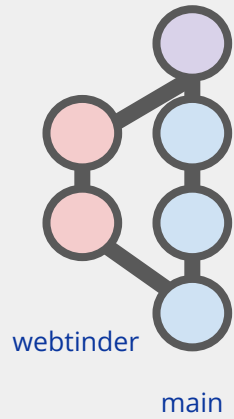


Git fetch

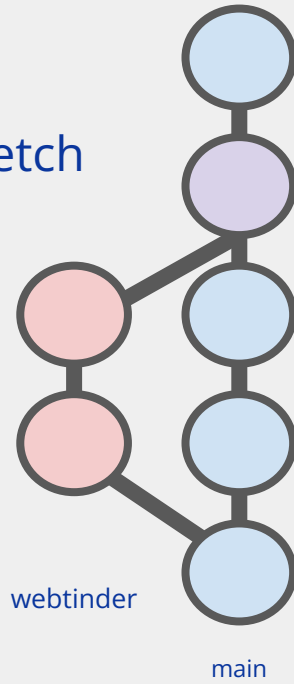


Remote Server (github)

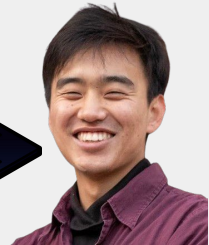
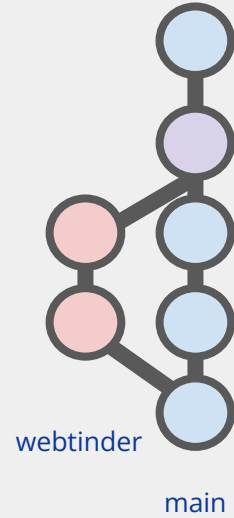


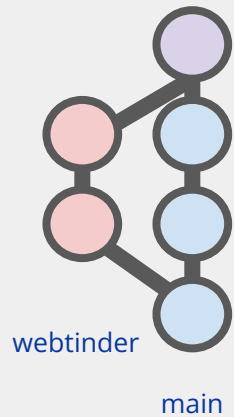


Git fetch

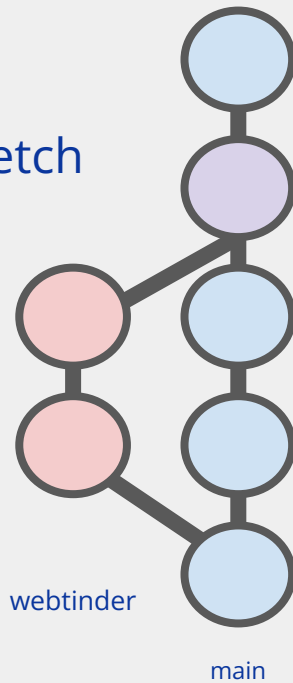


Remote Server (github)

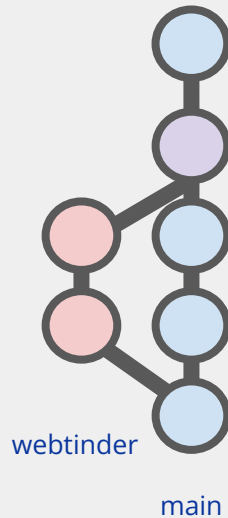


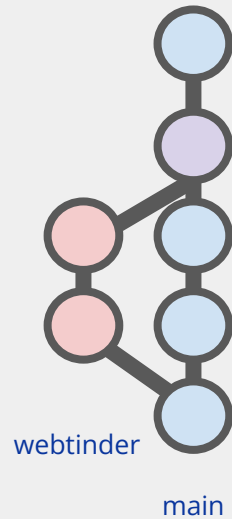
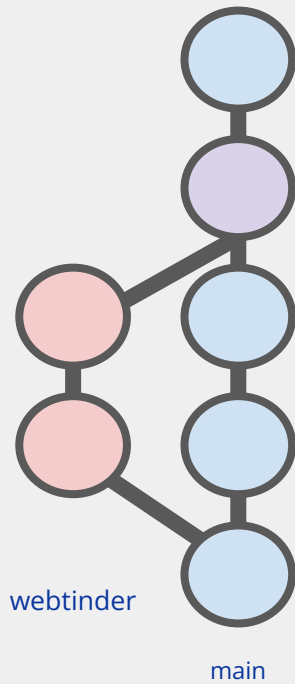
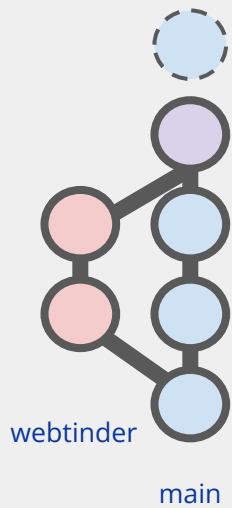


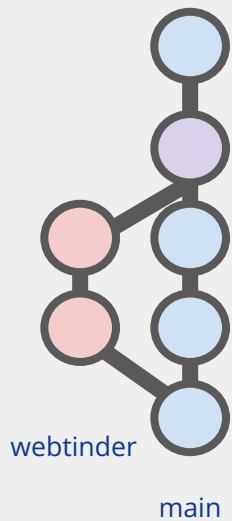
Git fetch



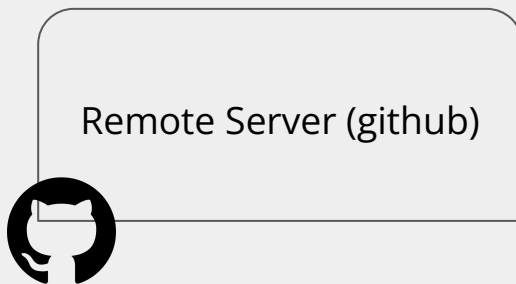
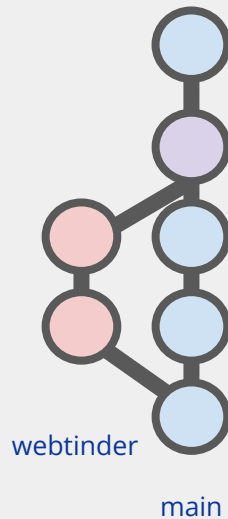
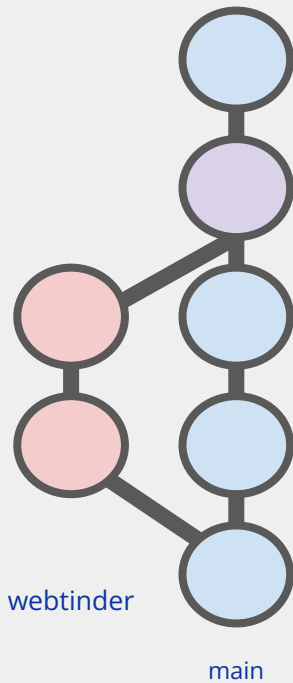
Remote Server (github)







git pull



Stay tuned!



git reset –hard

