viva berlenghi

Bio-210 Survival kit

1 - Git
2 - Conda
3 - Branches
4 - Commits
5 - Merging
6 - Local | Remote / Head
7 - Releases / Tags / Issues

All the things I wish I understood right away (plot twist : I didn't)

---

(1) Git

What is git?

→ code management tool

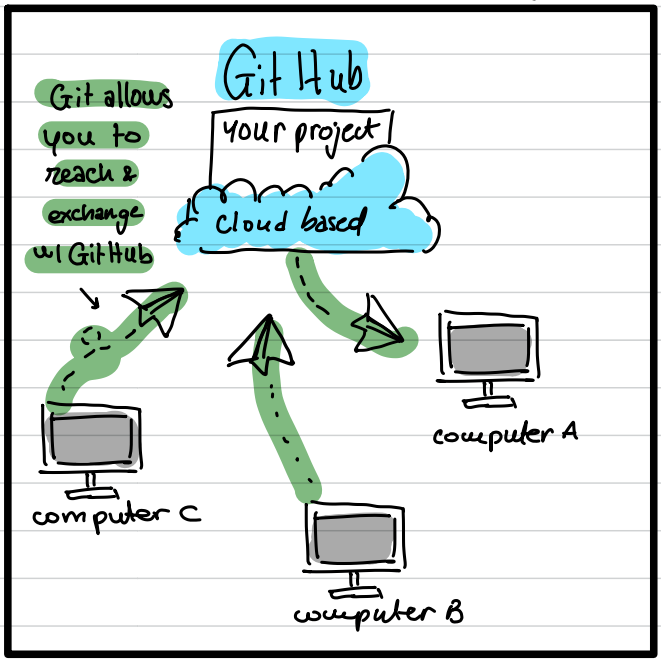
→ allows you to work w/ multiple people ↳ = 'with'

→ call it w/ 'git' command on terminal / bash

→ Allows you to use GitHub

What is GitHub?

→ code hosting platform for version control + collaboration (yay projects!)

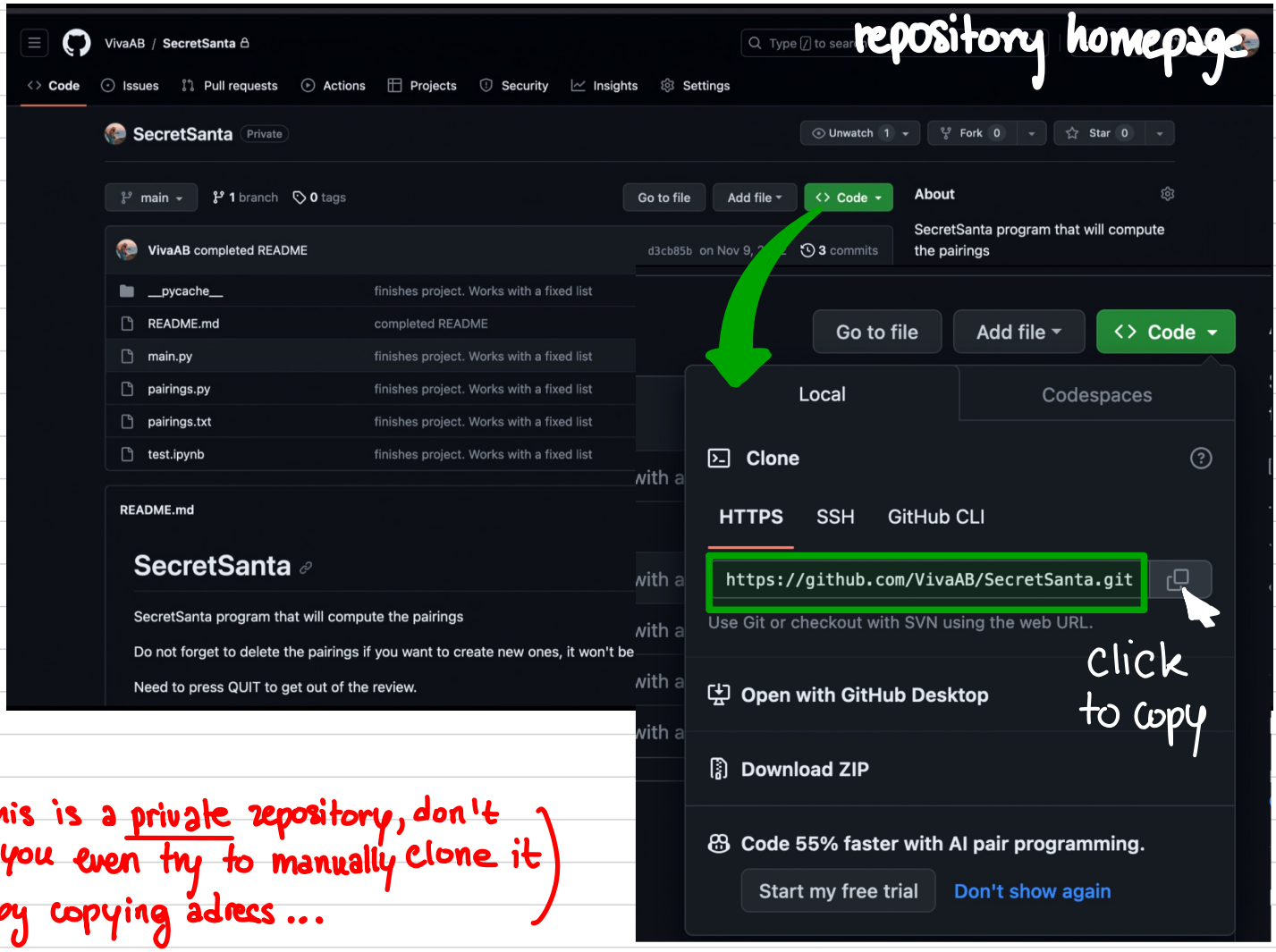


Check with yourself that you understand the difference between git & GitHub, ask a TA if not!

What is a repository?

→ contains all of your project's files and history

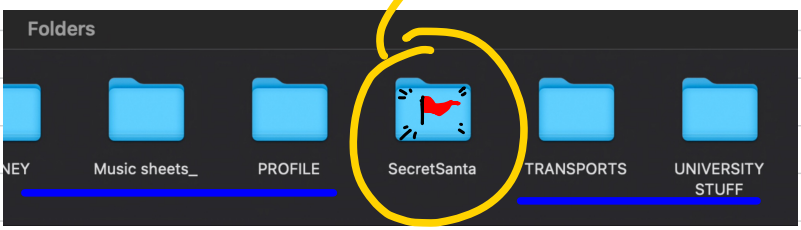
→ Add it to your computer w/ 'git clone <repo link>'



This is a private repository, don't you even try to manually clone it by copying adress ...

Can I see it on my computer?

KINDA...



NORMAL FOLDERS
NOT REPOSITORIES

→ it looks like every other folder → you can only access the content of the last branch selected

→ if you switch branch, the content will adapt

How can I tell it apart from regular folders?

Git knows!

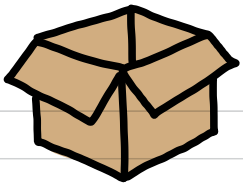
If you enter `git status` after going to your folder/repository path, you will either get

- fatal: not a git repository (or any of the parent directories): .git → regular folder

or

- on branch current branch name
  ↳ repository!

② What is CONDA ?



→ Conda is a **package management system**

↳ But **what does it do?** **Why do WE need it?**

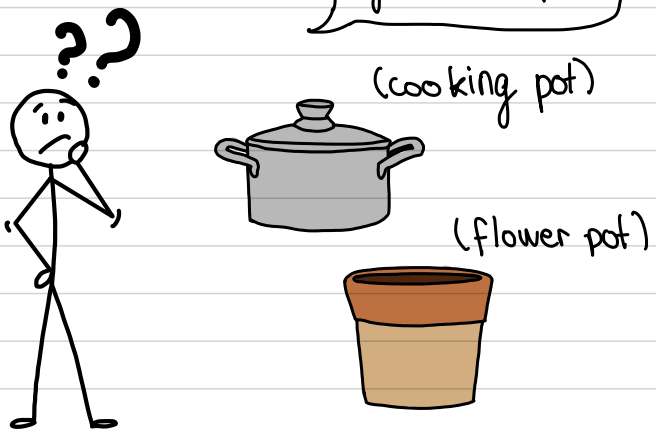
→ **package management**
- → allows to easily install, update and remove software packages (like numpy, matplotlib, and a... LOT more)
- → allows someone else to set up environment with all the tools required to run your code → .yml file

→ **environment management** → so you can work in isolated environments!

Why do we want that?

Imagine you are in this situation

and someone asks 'grab me a pot'



You end up pretty confused.

↓

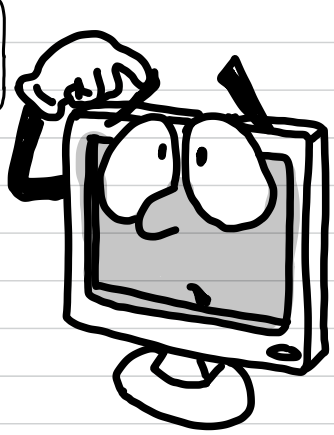
This happens to your computer w/ package!

Some different projects might need different versions of the same package, e.g. numpy 1.26.0 or 1.22.0

imagine you download both

Now when you say:

import numpy as np



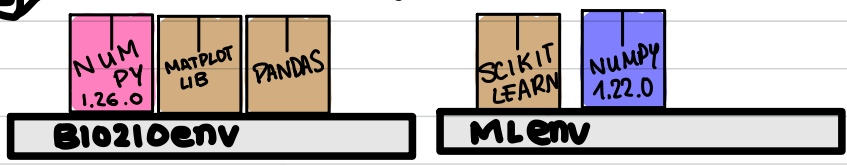
numpy 1.26.0

numpy 1.22.0

which one should I use?

what we do instead:

We create virtual environments and add specific packages to them:

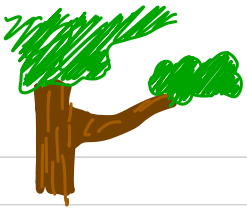


then we activate the environment we want to work in → `conda activate BIO210ENV`

↳ name of your environment

and we can avoid packages issues!

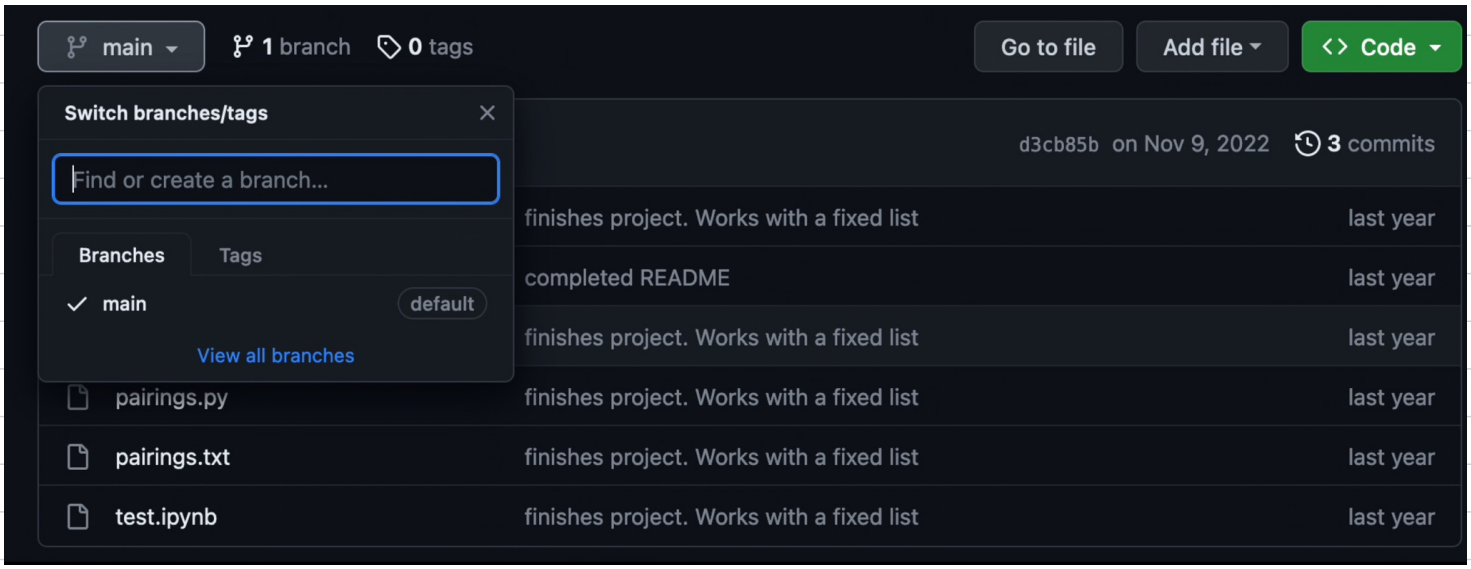
③ BRANCHES



What is a branch? → allows you to work on features, try fixing bugs or experiment in a safe way without messing up the project.

→ you always create it from an existing one!

find it on GitHub:



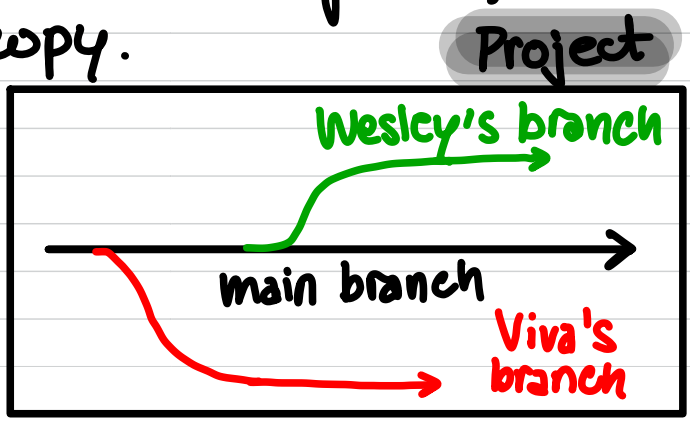
Why do we use branches?

→ allows you to **easily deal with conflicts** between code versions.

→ If you create your own branch you can be the only one working on it so nobody's work might mess up yours and vice versa

→ you create it from **another existing one**, kind of like if you made **a copy**.

* what is a branch supposed to contain when you create it?



Typical stuff we do with branches:

- see which branches are there: git branch

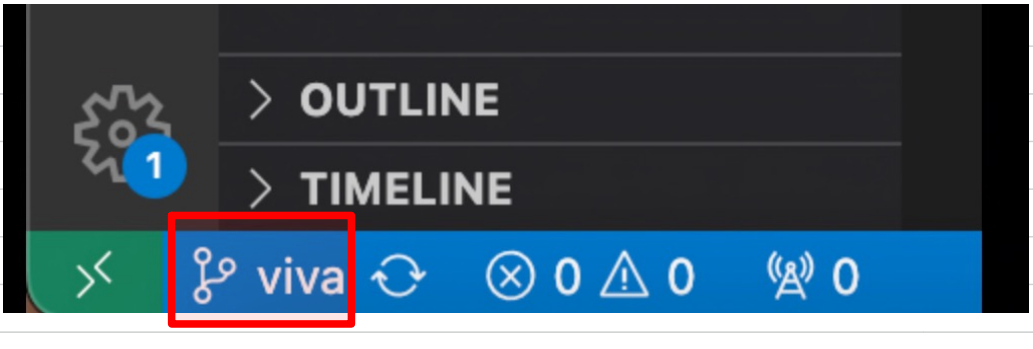
```
(base) MacBook-Air-Viva:ml-project-1-bytemeifyoucan vivaberlenghi$ git branch
  main
* viva
```

↳ branch you're on

- create branch from the one you're on: git branch branch name
- switch branch: git checkout branch name
- create + go on branch: git checkout -b branch name
- delete: first get out so checkout another one then git branch -D branch name

Why this makes sense?

- which branch am I on, aside from git branch:



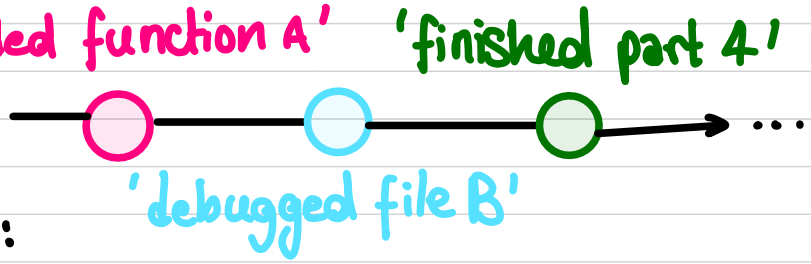
Bottom left corner on VSCode

4 COMMITS

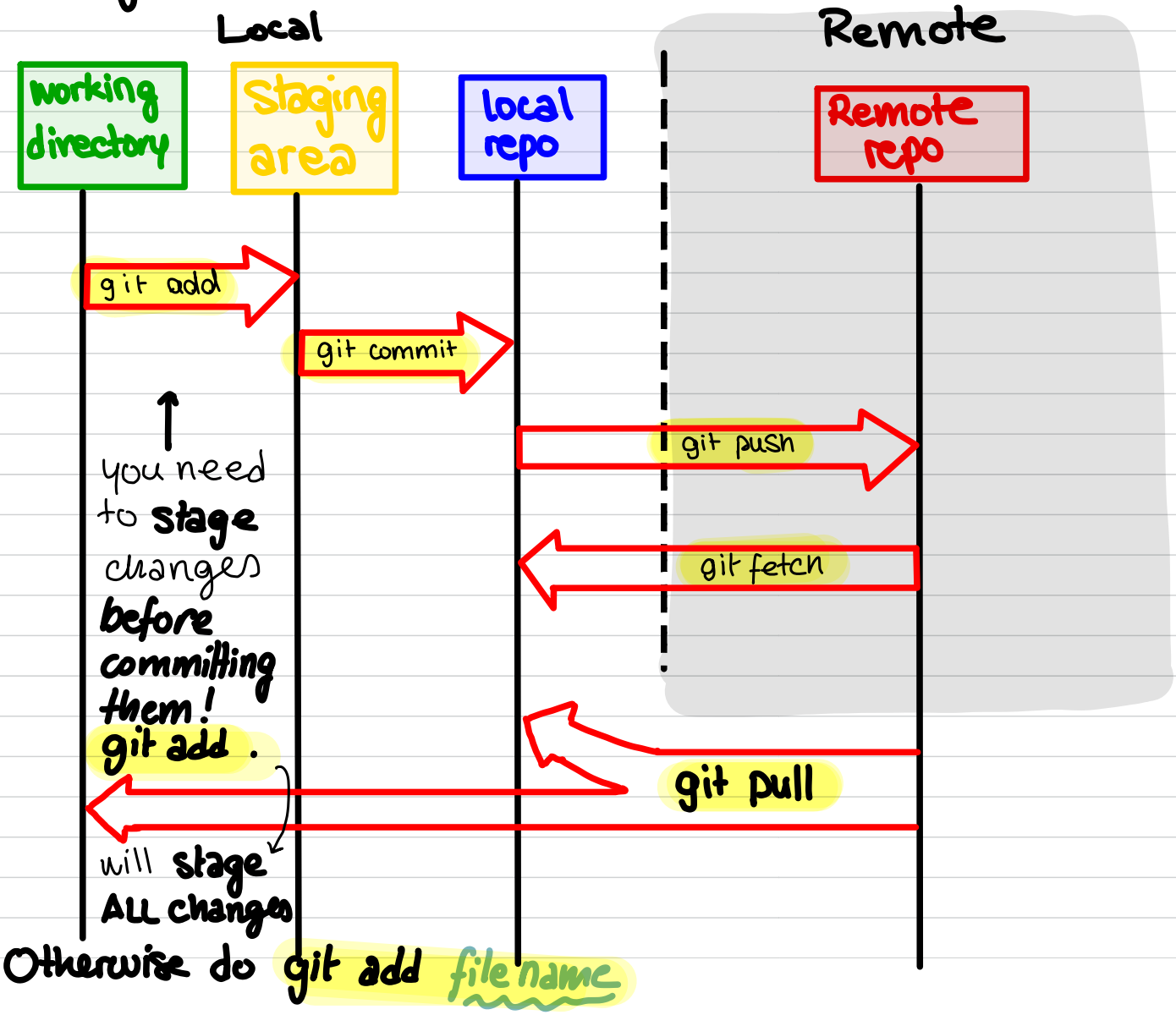
What is a commit? → **not a folder, nor a file.**

→ **You commit** when you are **satisfied** with your work and want it **in the repository**.
A bit like **ctrl+s** for the repo (ctrl+S will just save your work on your workspace)

It's like **a timepoint on your project history:**



let's go back to this:



Common stuff to do with commits:

- move to a specific commit: `git checkout commit name`
- committing: `git commit -m "commit message"`
- committing w/ co-autors:

```
$ git commit -m "Refactor usability tests.
>
>
Co-authored-by: NAME <NAME@EXAMPLE.COM>
Co-authored-by: ANOTHER-NAME <ANOTHER-NAME@EXAMPLE.COM>"
```

- sending commits to remote repo: `git push`
- see list of commits: `git log` ↴

```
commit 83c9564add70123878f817687491f9eec525092a (HEAD -> main, origin/main, origin/HEAD)
Merge: d3cb85b ac36e43
Author: Viva Berlenghi <92450252+VivaAB@users.noreply.github.com>
Date:   Sun Oct 15 19:32:04 2023 +0200

    Merge pull request #1 from VivaAB/workingonfeatureA

    add feature A

commit ac36e43be2c46e6fb4c536cbbbafac0db9608554 (origin/workingonfeatureA, workingonfeatureA)
Author: VivaAB <viva.berlenghi@epfl.ch>
Date:   Sun Oct 15 19:28:59 2023 +0200

    add feature A

commit d3cb85bdd928f0bf2dfd0b005a1f0e6e30ef828b
Author: VivaAB <viva.berlenghi@epfl.ch>
Date:   Wed Nov 9 11:24:10 2022 +0100
```

- go to the previous commit: `git checkout HEAD^`
  or 2 commits above that: `git checkout HEAD^^`

what if I want to go up 20 commits? 20 x ^ ?

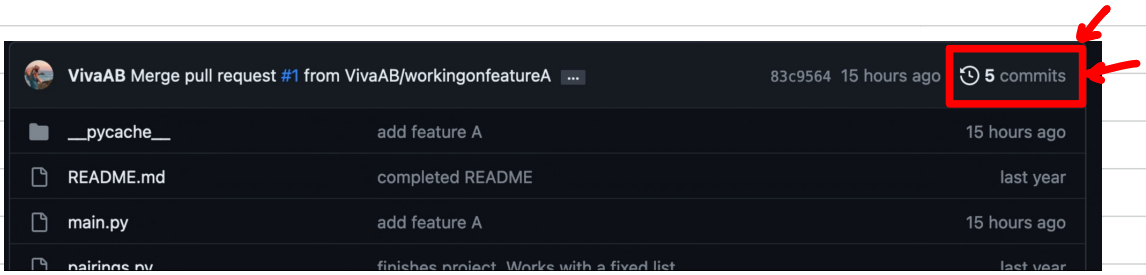
`git commit HEAD~20`

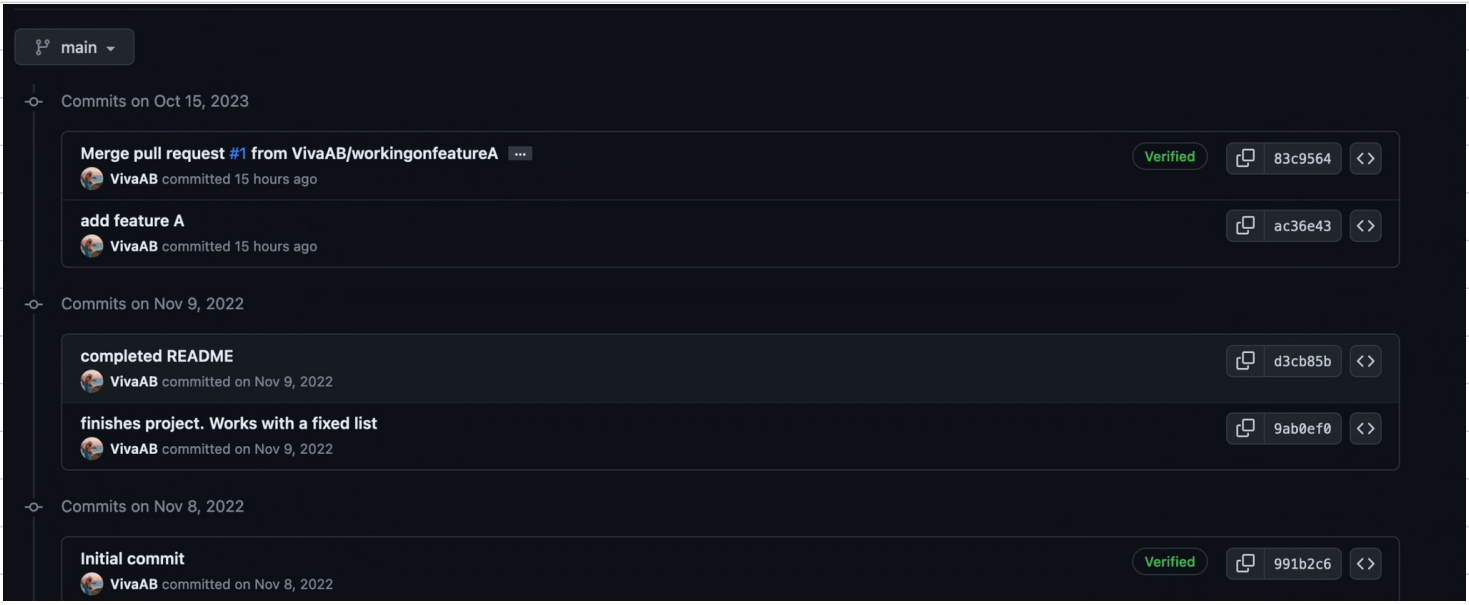
^ : move up 1 commit at a time

~num : move up a number of times

→ HEAD is where you are currently!
more about that later

What does it look like on GitHub?

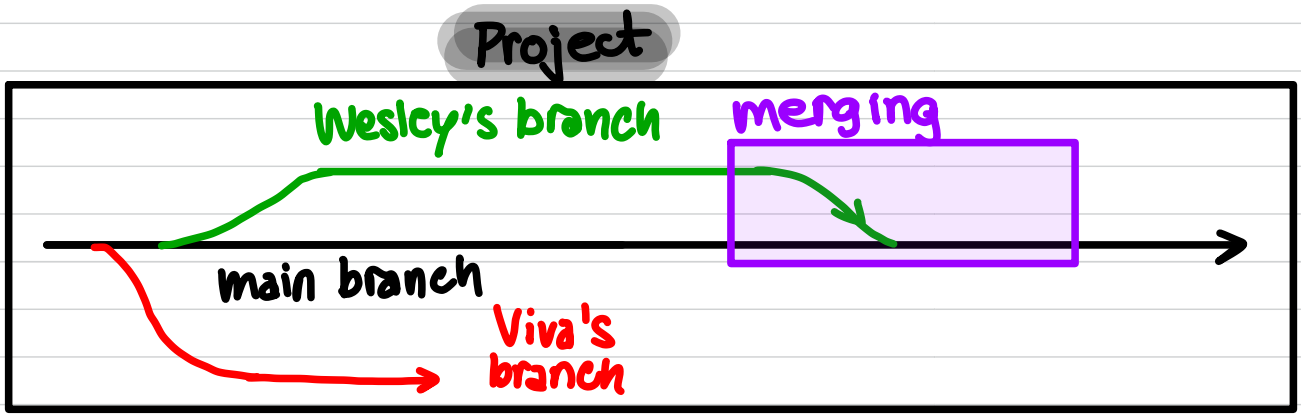




* Understand what is a commit and when to make one!

⑤ MERGING

What is merging? → incorporating changes from a branch into another one



↳ 3 branches:
- main
- branch A
- branch B

how do we do this?

After comitting changes to branch B → checkout main

↓

update it by pulling (on the files branch B changes)

↓

git merge branch B

↳ if the main branch didn't have changes between the moment you created the branch and when you merge

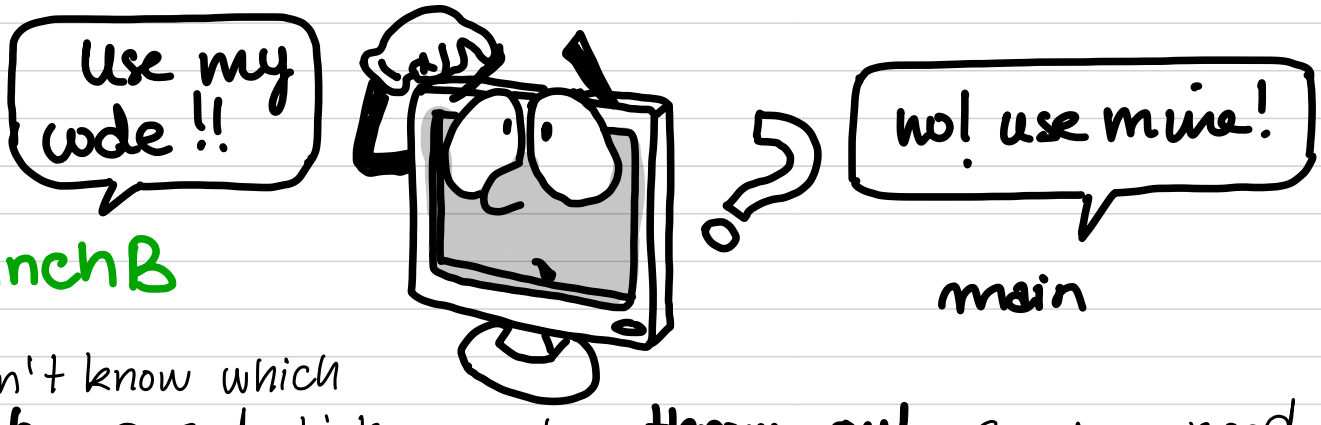
↳ you are good to go!

- stage changes `git add .`
- commit `git commit -m 'merge'`
- push `git push`

↓

if changes have been made on main since you created your branch you will get **conflict**

What is conflict? How do we solve it?



Git doesn't know which version to **keep** and which one to **throw out** so you need to **manually review** and **choose**

In VSCode

If the message I get when I try to merge branchB into main (I merged branchA into main after creating branchB from main)

```
(base) ➜  SecretSanta git:(main) git merge branchB
Auto-merging main.py
CONFLICT (content): Merge conflict in main.py
Automatic merge failed; fix conflicts and then commit the result.
(base) ➜  SecretSanta git:(main) ✗
```

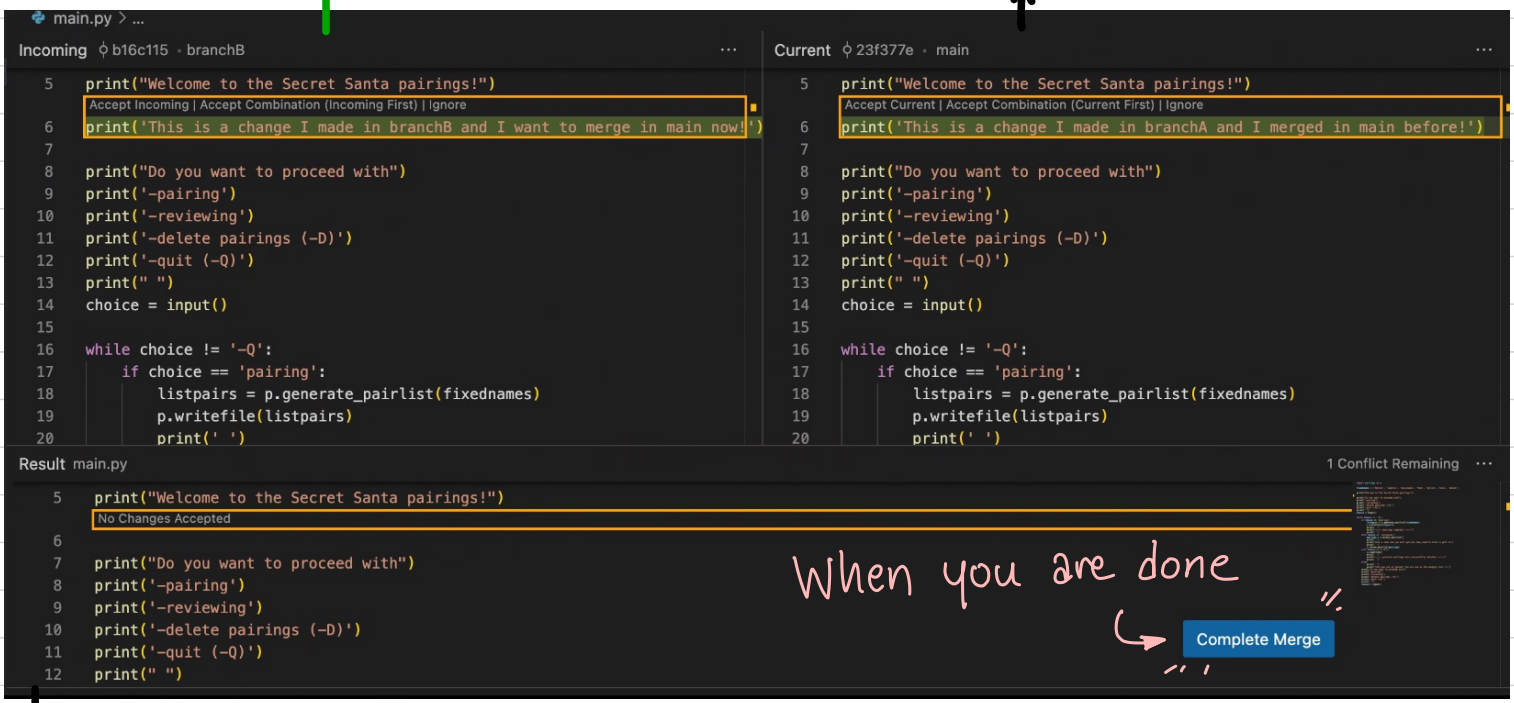
When doing so, the name of the files with conflict turn red when you click on the file:

with 1 click you can resolve the conflict! Or manually delete lines

```
5   print("Welcome to the Secret Santa pairings!")
    Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
6   <<<<<<< HEAD (Current Change)
7   print('This is a change I made in branchA and I merged in main before!')
8   =======
9   print('This is a change I made in branchB and I want to merge in main now!')
10  >>>>>>> branchB (Incoming Change)
11
12  print("Do you want to proceed with")
13  print('-pairing')
14  print('-reviewing')
15  print('-delete pairings (-D)')
16  print('-quit (-Q)')
17  print(" ")
18  choice = input()
19
20  while choice != '-Q':
```

For more detail → Resolve in Merge Editor

changes on branchB ↑ changes on main (since branchB was created)

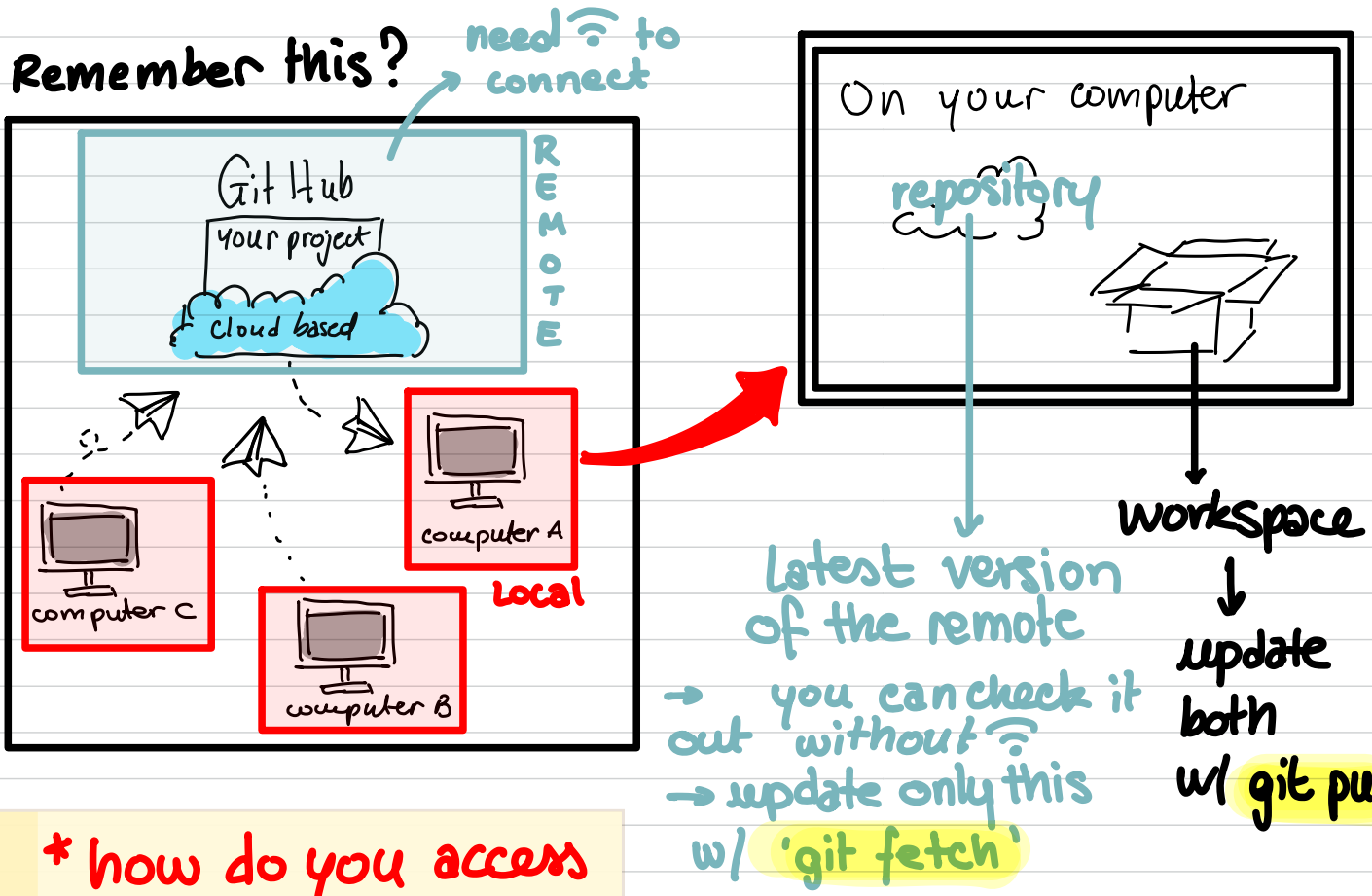


↳ resulting file after you accept one/both/none of the changes

Once you are done: stage, commit and push your changes

(6) Remote / Local / head

Remote - local what do you mean?



* how do you access your repo on your computer?

* do you understand the difference between remote & local work?

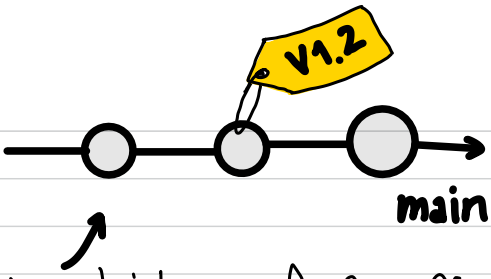
What is HEAD?

→ symbolic name for the **last commit** you **currently working** after

→ unless you **checkout a commit** it will be **attached to the branch** you are working on & on its **latest commit**

→ **Detaching HEAD <=> attaching it to a commit instead of a branch**

7 Releases / tags / issues



What is a release/tag?

TAG: → a **specific commit** in the version history of a repo, marked w/ a **tag**, typically a meaningful **label**

→ **used for milestones (e.g: a release)**

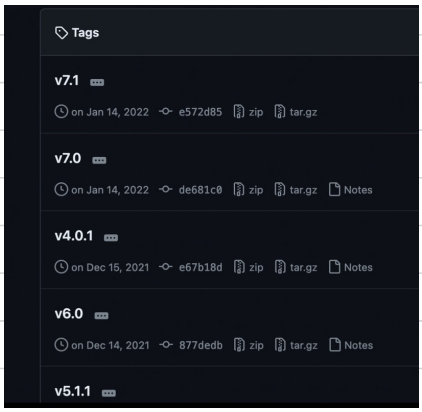
RELEASE: → a version of the code that is **'finished'**
→ may correspond to **a specific tag**

Why bother?

→ this will allow you **to clearly show TAs** where they should look **to evaluate your code** while you can **continue adding features!**

What does it look like for a TA correcting:

→ **after adding a tag on GitHub**



→ **You can checkout the commit associated to the tag very easily!**

```
commit 01b79c4a2effa2f0e3d1b2e26ae35b7b7d2e019c (HEAD, tag: v5)
Author: [redacted]
Date:   Tue Nov 22 21:48:04 2022 +0100

    test for list

commit 0a8d9365a5a43bfdc2a4f30f1616214dc3a3801e
Author: [redacted]
Date:   Tue Nov 22 21:44:42 2022 +0100

    little changes/test

commit f081a15226370701c4ee4beccda58771ac3b70f6
Merge: bfb9c7f 651df16
Author: [redacted]
Date:   Mon Nov 21 15:47:54 2022 +0100

    last version main

commit 651df16fa5dfd351ce7a438ec43fe98350869aa1
```

→ **then I can do**

git checkout v5

and immediately access it

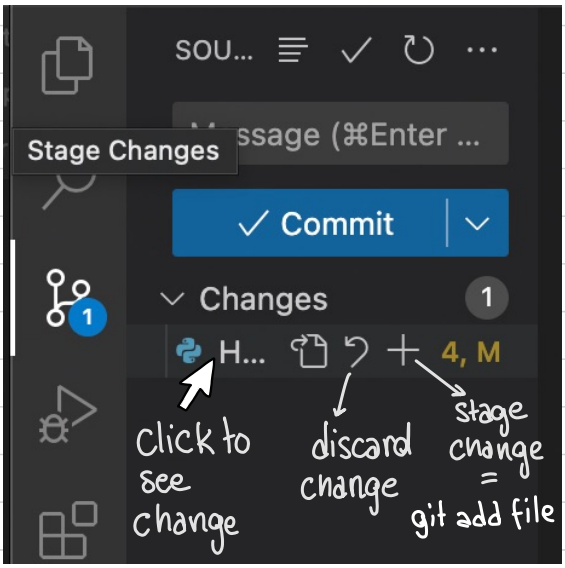
What is an issue?

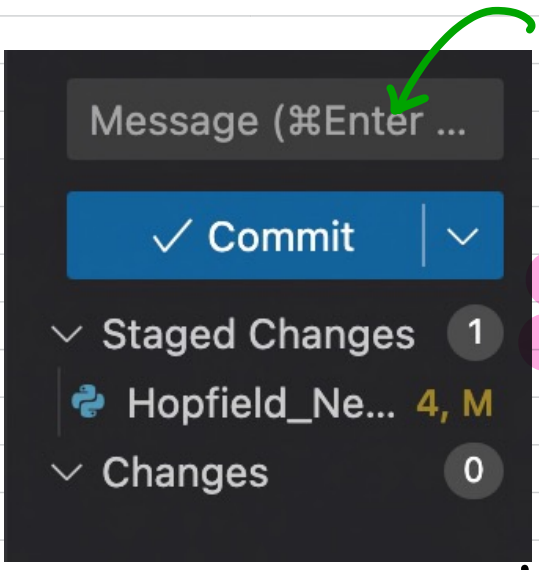
→ allows you to add to-dos and signal problems to your teammates and assigned TA.

⚠ this is how you will communicate w/ your TA so check it out and use it for your code-related problems

Can I do some of this in a more visual way?

YES! While you usually should use command lines, VSCode allows you to visualize some steps of the workflow:





you can write your commit message & commit the staged changes

then press again to push your changes

⚠ You need to link your GitHub account to VSCode to be able to sync changes

→ look at the bottom left side if 👤 is connected to your GitHub account