Free Software and Version Control 101

A introductory course on FOSS, git, curl and web integrations

J. Rodrigues F. Carvalho R. Gama

HackerSchool

Section 1

Version control?

What is version control?

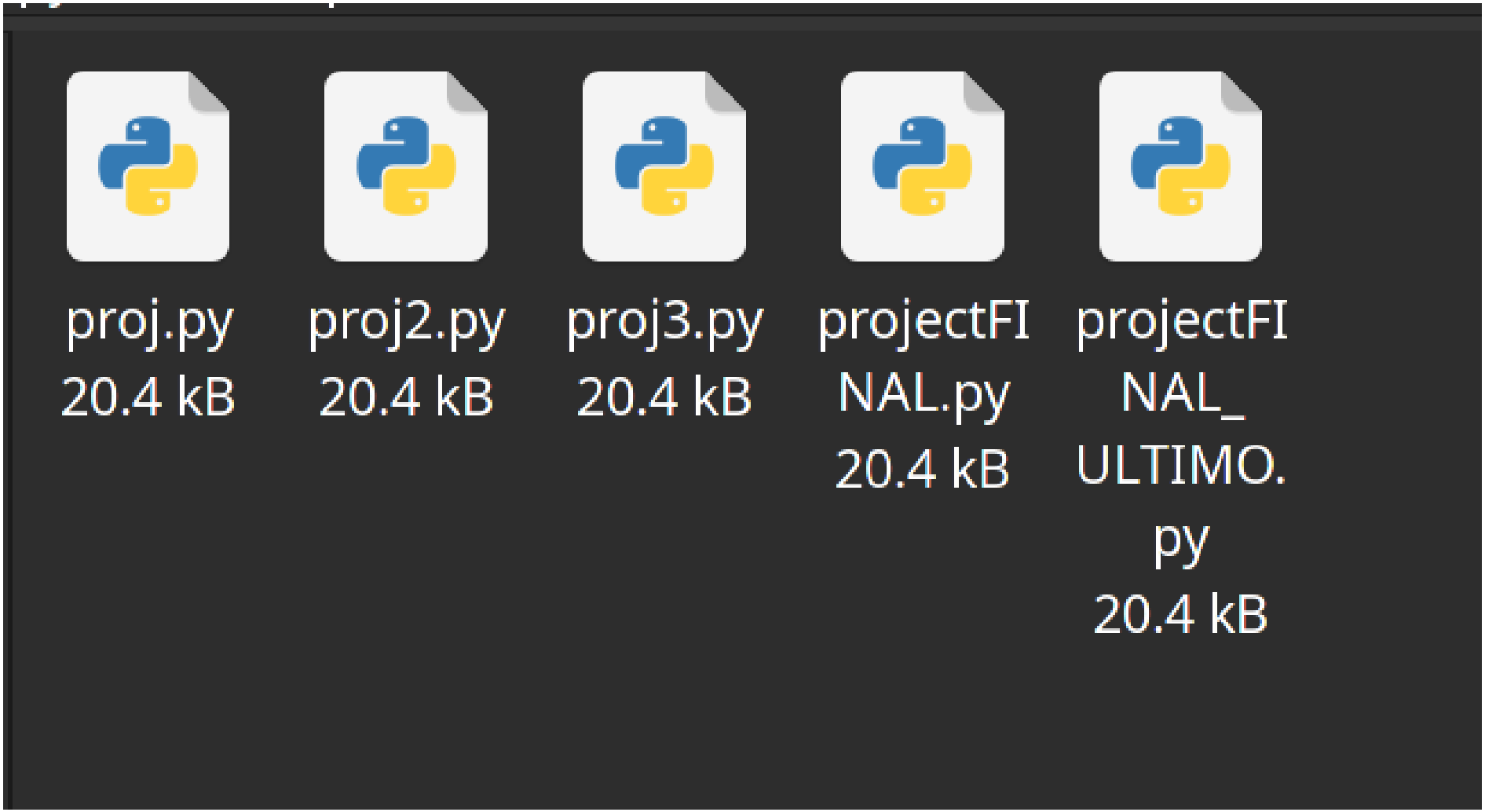
Version control is the practice of managing and documenting *data* (code, schematics, etc.) iterations.

It is particularly important in our context of Free and Open Source software, as a careful documentation of alterations between versions and the ability to inspect older or deprecated sources can make issue resolution and feature integration much more agile.

git?

Git is a version control software created by Linus Torvalds (which also created the Linux Kernel). Its free software under the GPL v2.0.
Git allows cloning, pulling, pushing, etc. of data stored in git instances.

It has happened to all of us!



Time to ditch this. . .

For something waaaay better

```
* commit eebc2e011fa88bfae93e8105e665fa5873499569 (HEAD -> master, origin/master)
| Author: Francisco Carvalho <franciscojcarvalho@tecnico.ulisboa.pt>
| Date:   Thu Oct 13 16:42:39 2022 +0100
|
|     Database changes (to be implemented in backend)
|
* commit bf2162b4080243bb4458f916319a53ebfa3d9253
| Author: Francisco Carvalho <franciscojcarvalho@tecnico.ulisboa.pt>
| Date:   Sat Oct 8 19:36:46 2022 +0100
|
|     Login system bypass as it's not needed. LF endings
|
* commit bf5c4c51b9fba1ffdc18b44abf24491d7a1bf6c4
| Author: Francisco Carvalho <franciscojcarvalho@tecnico.ulisboa.pt>
| Date:   Fri Oct 7 14:32:27 2022 +0100
|
|     .desktop file and autorun script
|
* commit 0b9bfc255efc9fc897189b46885acc7d51251e2c
| Author: Francisco Carvalho <franciscojcarvalho@tecnico.ulisboa.pt>
| Date:   Fri Sep 30 21:01:59 2022 +0100
|
|     Code cleanup
|
* commit 764c31a6c26b3ef19e8921895cbeb50cc271920f
| Author: Francisco Carvalho <franciscojcarvalho@tecnico.ulisboa.pt>
| Date:   Fri Sep 30 20:51:04 2022 +0100
|
|     Bug: week number stuck at 1
|
```

• schema.sql:9: CREATE TABLE IF NOT EXISTS etcs_users(

```
 9      course varchar,
10      primary key (ist_id)
11  );
12  CREATE TABLE IF NOT EXISTS etcs_subjects(
13      subject_id numeric,
14      term varchar,
```

```
 9      course varchar,
10      primary key (ist_id)
11  );
12
13  CREATE TABLE IF NOT EXISTS ects_work(
14      identifier numeric,
15      name varchar,
16      primary key (identifier)
17  );
18
19  CREATE TABLE IF NOT EXISTS ects_extra_activity(
20      identifier numeric,
21      description varchar,
22      type numeric,
23      foreign key (identifier)
24          REFERENCES ects_work(identifier),
25      primary key (identifier)
26  );
27
28  CREATE TABLE IF NOT EXISTS etcs_subjects(
29      subject_id numeric,
30      term varchar,
```

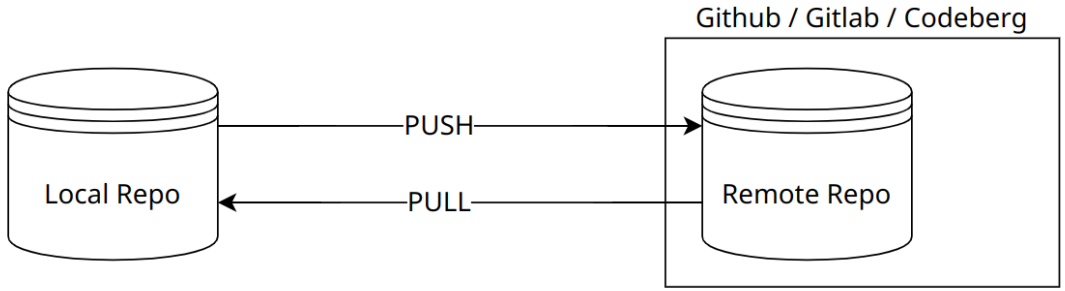
• schema.sql:37: CREATE TABLE IF NOT EXISTS etcs_subjects(

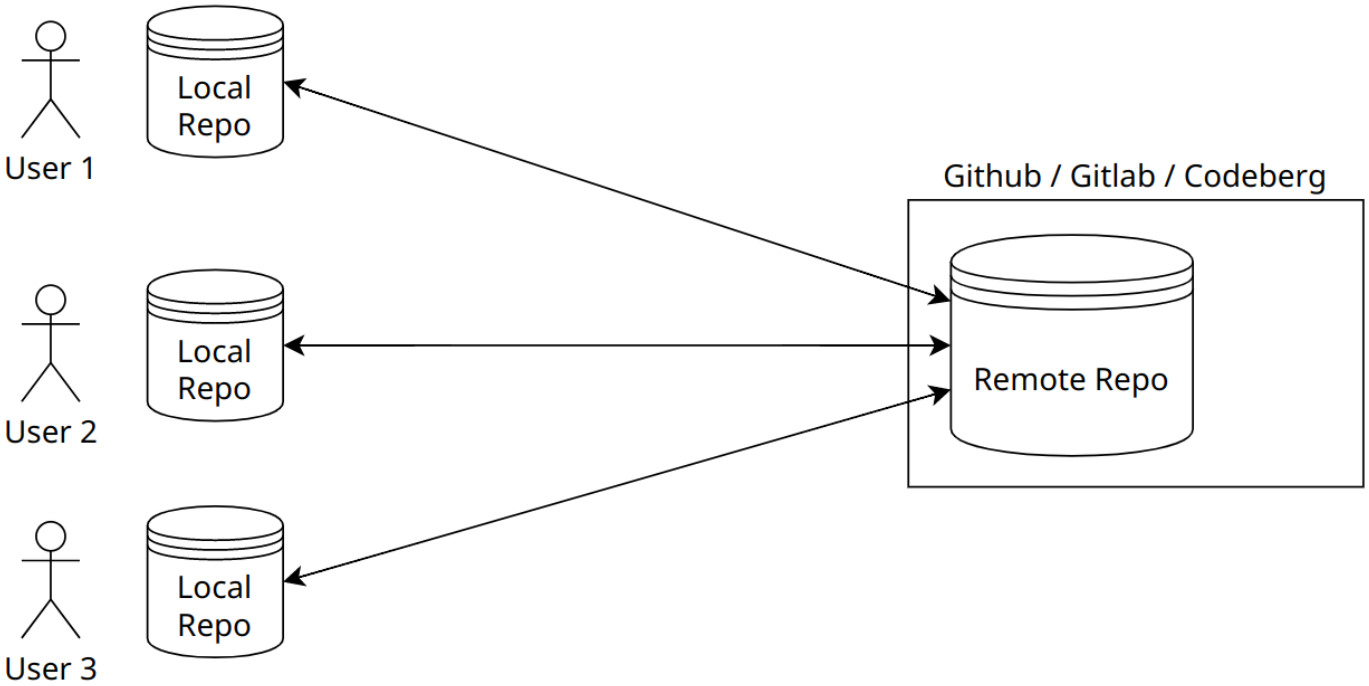
```
21      subject_ta numeric,
22      subject_t numeric,
23      subject_period numeric,
24      primary key (subject_id)
25  );
26  CREATE TABLE IF NOT EXISTS etcs_enrolled_in(
```

```
37      subject_ta numeric,
38      subject_t numeric,
39      subject_period numeric,
40      foreign key (subject_id)
41          REFERENCES ects_work(identifier),
42      primary key (subject_id)
43  );
44  CREATE TABLE IF NOT EXISTS etcs_enrolled_in(
```

• schema.sql:71: CREATE TABLE IF NOT EXISTS ects_academic_terms(

Remotes


Github / Gitlab / Codeberg
Local Repo
PUSH
PULL
Remote Repo


User 1
Local Repo
Github / Gitlab / Codeberg
User 2
Local Repo
Remote Repo
User 3
Local Repo

Section 2

Let's Start!

We will teach you how to work with it via the command line, but there are tools, like VS Code that provide the same functionality via a graphical interface

Setting up your git/GitHub environment

1. Create a GitHub account (using your institutional e-mail is often valuable).
2. Get the `git` and `github-cli` packages (this last one is optional).
3. Log in in github-cli (or if you prefer not to use it setup an SSH Key/ Personal Access Token)
4. Configure user in git with

```
git config --global user.name "@user.name"

git config --global user.email @user.email
```

Setting up a GitHub repository

1. Go to https://github.com/new

   (it's easier if you check "Add a README file")

After you've created your repo you need to download a local copy to do your work

2. Use the command `git clone` to do that:

   ```
   git clone <url_of_your_repo>
   ```

You should now have a new folder with a copy of the repository you've just created

Setting up a GitHub repository

If you are using a SSH key or PAT it is important to use the correct URL format

- HTTP URL
  - https://github.com/Joao-Ex-Machina/Tree-Network-Client
- SSH URL
  - git@github.com:Joao-Ex-Machina/Tree-Network-Client

Section 3

Your first commit!

Git Status

This command alows you to view the state of your project (repo)

```
francisco@archboxSigma:~/repos/studyTracker$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   main.py
        modified:   templates/dash.html

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        README.md
        __pycache__/
        app/
        config.ini.old
        diagramaEA.png
        ects_weeks.py
        etcs.db
        populate.sql
        static/favicon.svg
        templates/timer.html
        test.sql

no changes added to commit (use "git add" and/or "git commit -a")
```

Add

- When we want the git log changes made to a file `git add <file_path>`
- When the file hasn't ever been tracked add tells git to start to
- This command only selects the files/modifications, it **does not** commit

```
francisco@archboxSigma:~/repos/studyTracker$ git add main.py README.md
francisco@archboxSigma:~/repos/studyTracker$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   README.md
        modified:   main.py

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   templates/dash.html
```

Figure 2: Git Add

Commit

- To commit (record the selected changes to the history) we use the command `git commit -m "commit message"`

```
francisco@archboxSigma:~/repos/studyTracker$ git add main.py
francisco@archboxSigma:~/repos/studyTracker$ git commit -m "minor comment"
[master a6b509e] minor comment
 1 file changed, 2 insertions(+)
francisco@archboxSigma:~/repos/studyTracker$
```

Figure 3: git commit

Push

- You can push your history to a remote repo using the command `git push <name_of_remote_repo>`
  - Usually when using Github this remote is called **origin**
- You can even have multiple remotes!
  - to add one use `git remote add <name> <url>`, as seen in the curl part of this presentation

```
francisco@archboxSigma:~/repos/studyTracker$ git push origin
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 367 bytes | 367.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
remote: This repository moved. Please use the new location:
remote:   https://github.com/SrFrancisco/tracker.git
To https://github.com/SrFrancisco/etcs.git
   eebc2e0..a6b509e  master -> master
```

Figure 4: git push

Frontends (Vscode)

Repositories and actions (*)

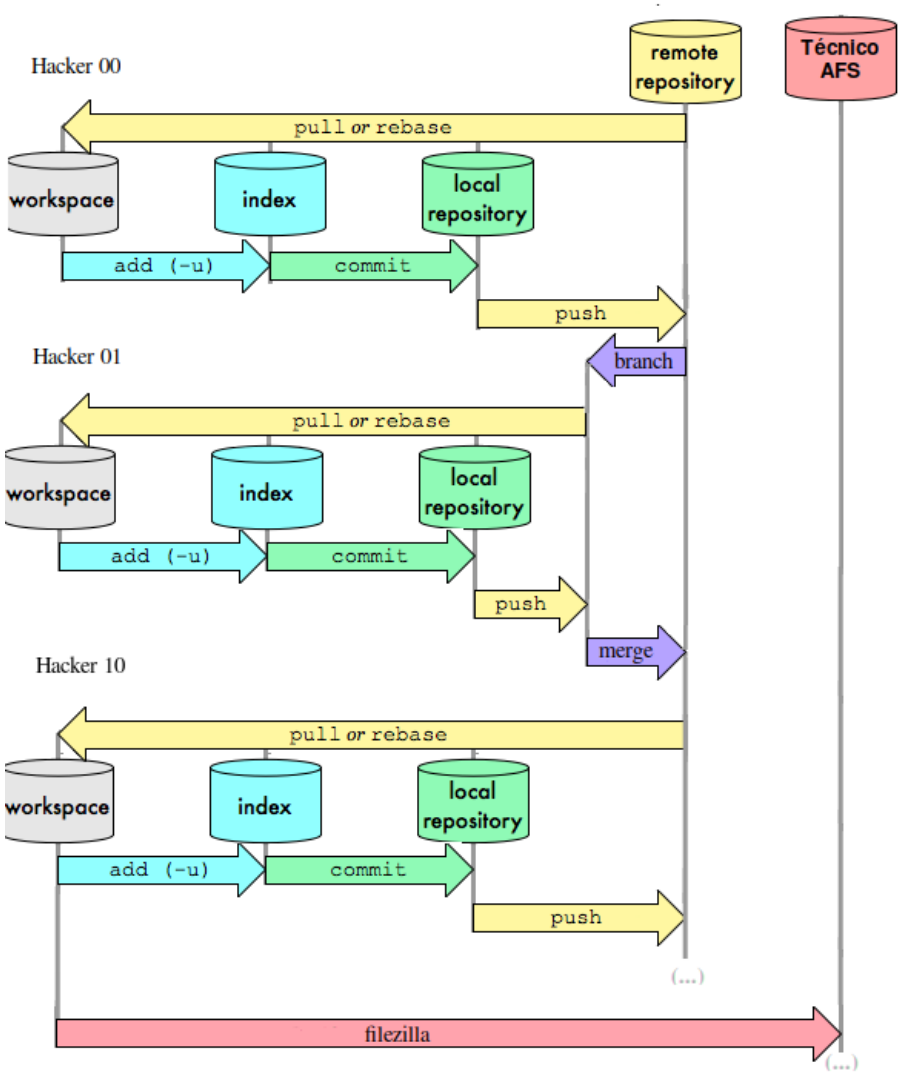


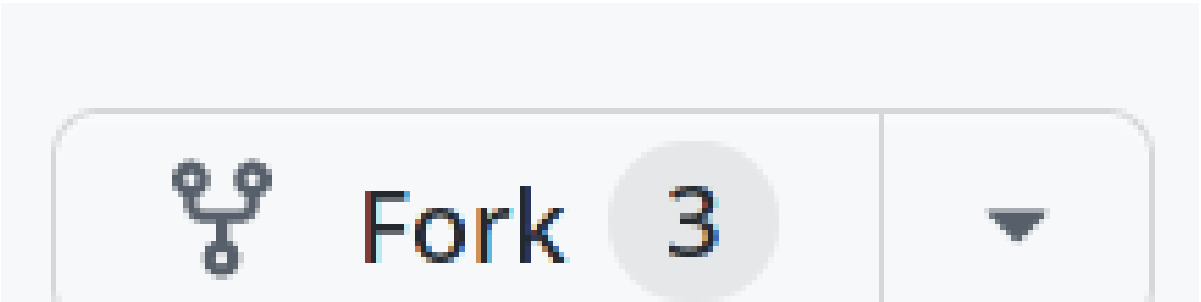
Figure 5: Git/GitHub flow of the hackerschool.io repository

Branching

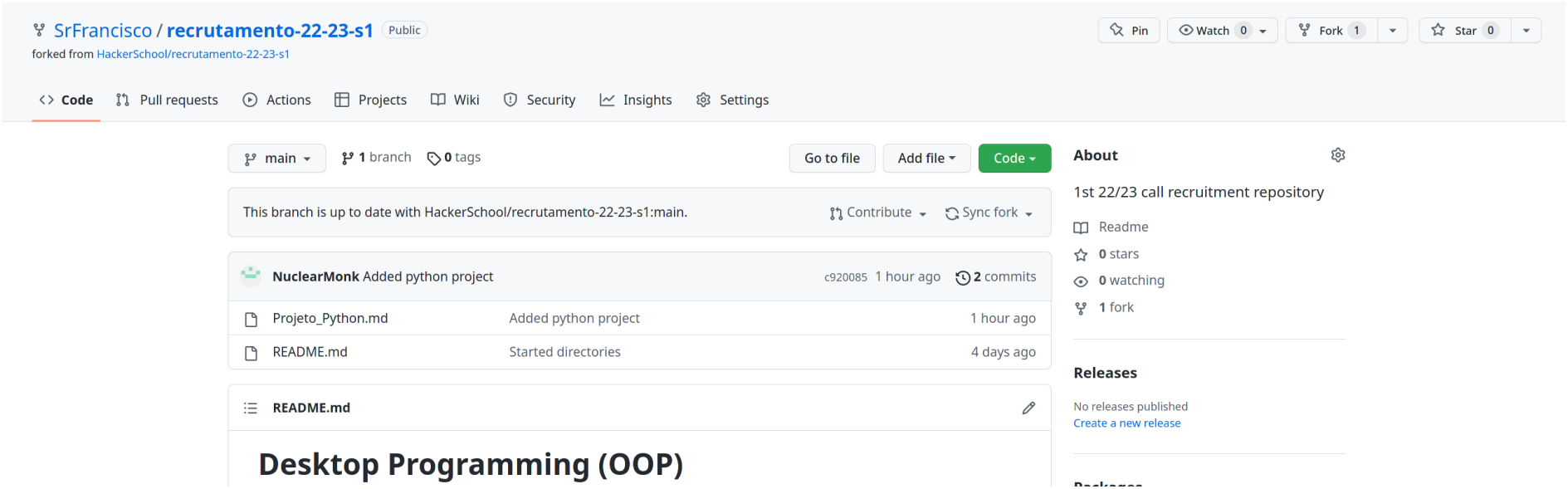
- Branching is a elementary tool that is part of git
- A branch is an alternative streamline from main
  - It is used by collaborators that normally want to push some code that can break the master's features or that has yet to be fully tested
  - A branch can be switched into using `git checkout @branch` or can be created and switched into by adding the `-b` flag to the previous command
- In medium-small projects, branching may be enough

Forking (*)

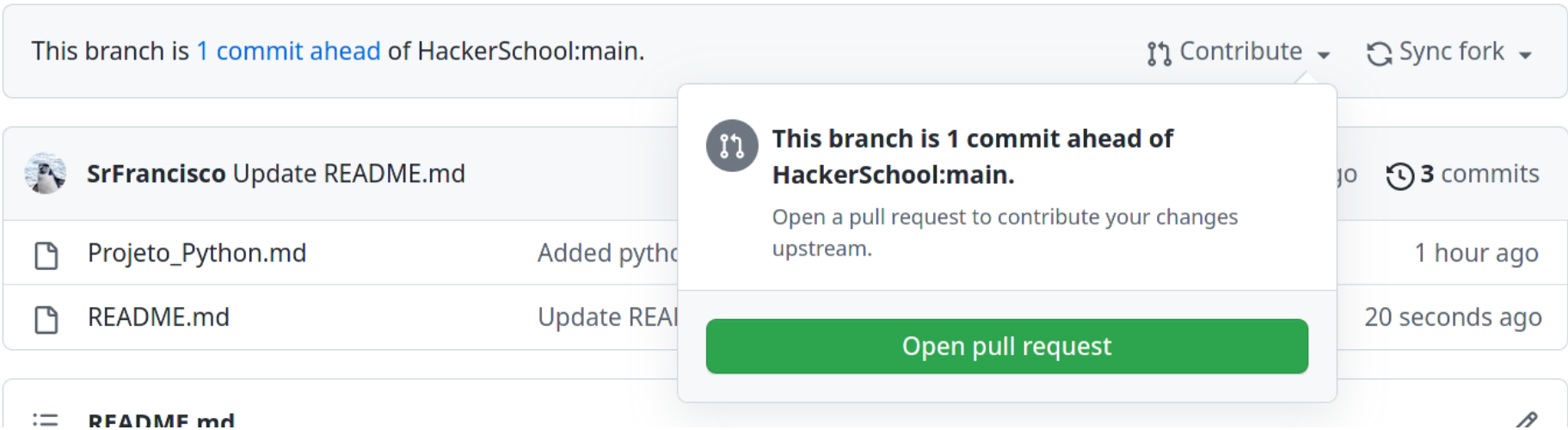
- !!! Note: This is a feature of some git instances (GitHub, GitLab, etc.), but not git itself, therefore we will not use the `git` terminal package in this section
- Nonetheless forking is an important collaboration tool, as it allows you to make your own private copy of an exciting repository
- This means you can work freely, without pushing "trash" to the main repo
  - You can even start your own version of the project!
- Once you're ready to merge your changes into the main repo you can open a **pull request**
  - This is one way of contributing code to a public repo if you aren't a contributor (=have write access)



Pull Requests



When we want to merge with the original repo we open a pull request



Conflicts (*)

Sometimes, when you and another collaborators make a change to the same line, GitHub's auto-merge feature cannot fix it.

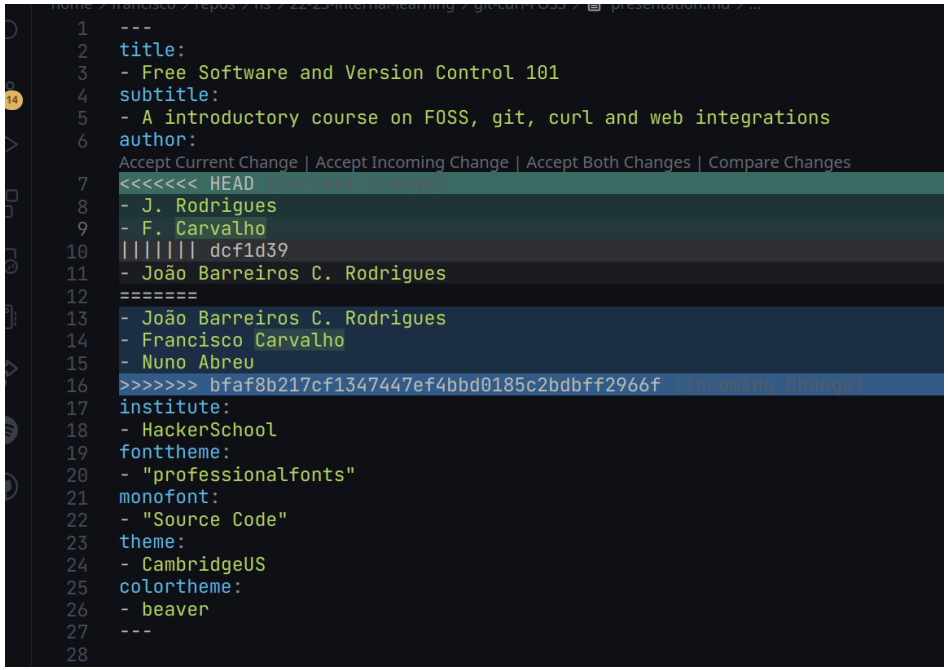


Figure 6: A wild conflict appears (while making this presentation)

You will have to manually fix it by choosing which lines to keep from each branch! Trivial!

Section 4

Tying our work with freedom

On FOSS and CC

The first written document that described FOSS as a trend among hackers, programmers, engineers, etc. was the GNU Manifesto. This file also gave the philosophical foundations for the Free Software Foundation Network, and FOSS worldwide.

In the context of HackerSchool FOSS is a core principle. Initiated in the administration of 22-23, HackerSchool has embraced free alternatives such as *GNU/Linux distros, FreeCAD, Jitsi, Signal*, allowing hackers to walk a path that is flexible, secure, and overall hacker-y.

Since we benefit from this communal effort it is only fair that we also contribute to the greater good, therefore all HackerSchool code and documents are non-proprietary.

The two software architectures

- **The Cathedral**

  We can apply this architectural decision to both:

  - **Free Software**

    The source code is centralized in an organizational environment, however it is released with any main binary. You will rarely look to the code in development.

  - **Proprietary Software**

    The source code is never released, it is kept confined within the Corporations walls. The binary is spewed out of it, but you will have to reverse it to understand what it does!

The two software architectures

- **The Bazaar**

  The Bazaar is characteristic of free software.

  In a Bazaar architecture the source code is visible at all times, being normally communal projects, made of differently written modules. Everything is a node on a de-centralized tree!

  This architecture as the advantage of going according to Linus's Law:

  "given enough eyeballs, all bugs are shallow"

Licenses

	Public Domain	Permissive	Copyleft	Non-Commercial	Proprietary
Is the source code available?	Yes	Yes	Yes	Yes	No
Copyright and authorship claim	No	Yes	Yes	Yes	No
Can I copy it	Yes	Yes	Yes	Yes	No
Can I distribute it?	Yes	Yes (under the same license)	Yes (under the same license)	Yes (under the same license)	No
Can I modify it?	Yes	Yes	Yes	Yes	No
Can I sell it? (modified or unmodified)	Yes	Yes	Yes	No	No
Can I sublicense the modified code?	Yes	Yes	No	Yes	No
Software License Example(s)	The Unlicense CC0	Apache License BSD License MIT License	GNU Public License (GPL) Affero GPL	Aladdin Free Public License	Proprietary License
Other Documents License Example(s)	CC0	CC-BY	CC-BY-SA	CC-BY-NC	Proprietary License
Software/media/file example	"Equilibrios líquido-vapor de componentes da aguarrás para destilação multicomponente"	"Big Buck Bunny"	Linux Kernel GNU utils 99.9% of the code I make	GhostScript	Michaelsoft Bimbows