IMPERIAL

Good programming practices and code management

27 Jan 2025

Dr Mostafa Safaie (m.safaie@imperial.ac.uk)

Agenda

Intro to version control
Intro to git
Common git workflows

GitHub interface
Intro to vs-code

Reviewing a project
Code/data management
Good coding style

Collaborating on a mini project
Reviewing your contributions

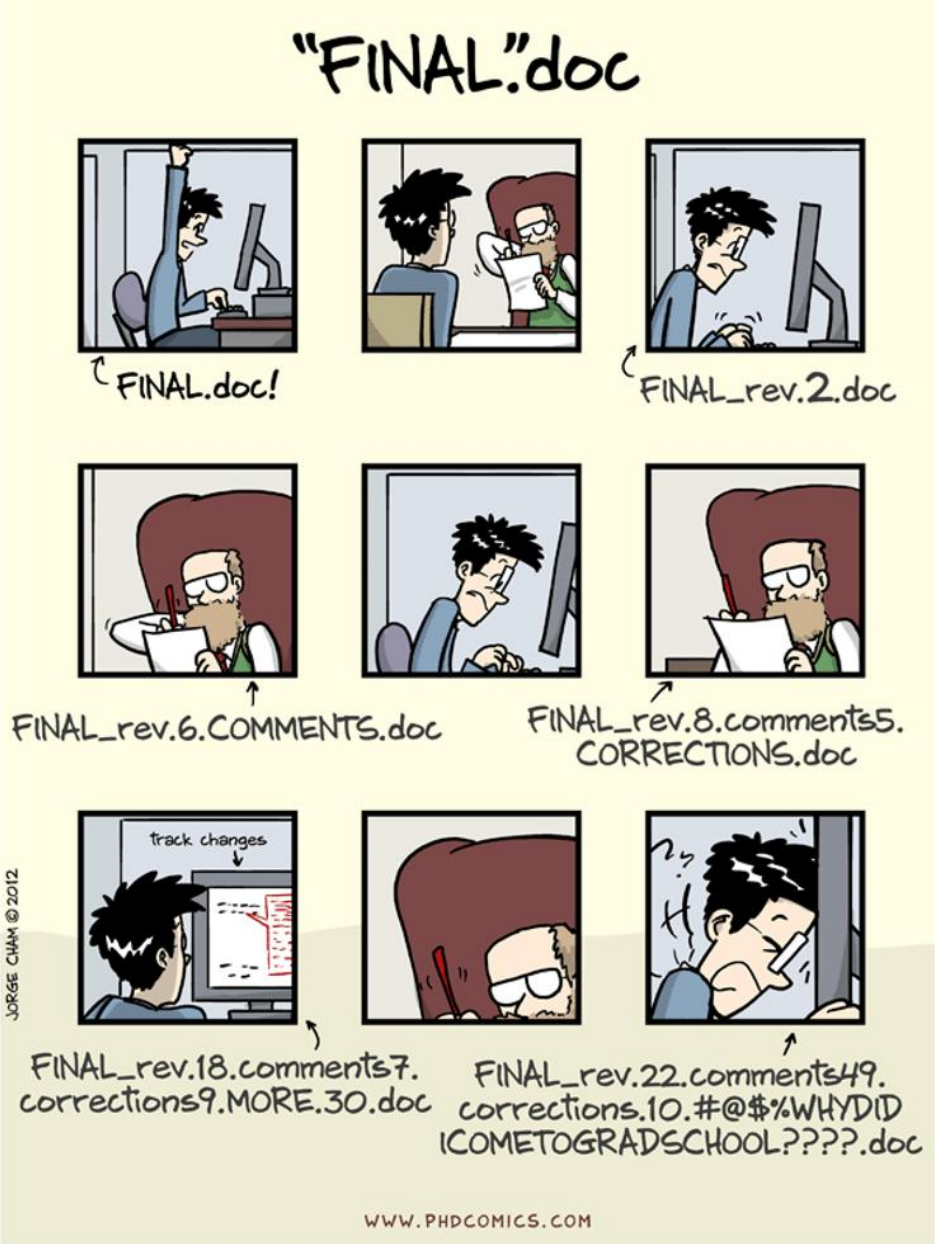
Intro to version control

‘Git’

Why bother?

- Clunky at first
- More complicated workflow

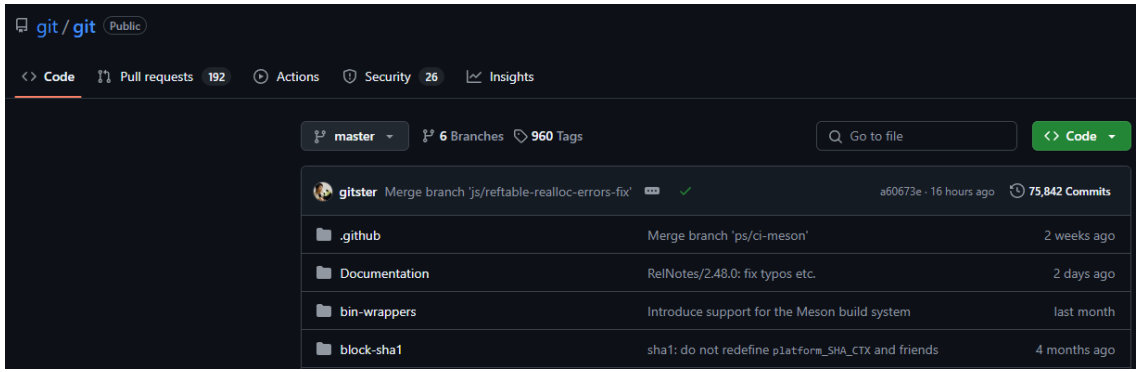
- Save effort / no losses — an unlimited *Undo*
- Teamwork
- Peace of mind
  - Can always revert
- Reproducible
- Imposes discipline



Git
What is it?

Version control (method of tracking & managing changes to –text– files)

- Open source
- Track changes you make to code or other documents (e.g., LaTeX)
- Check older versions, or look at changes



Collaboration

- Multiple developers
- Track individual user changes, and assign tasks

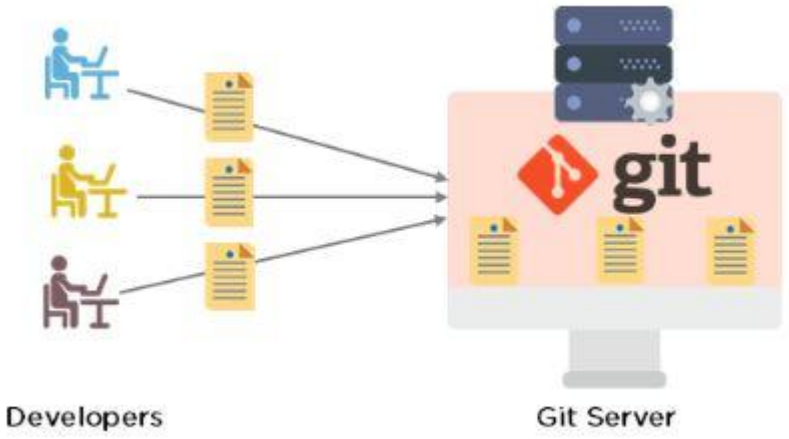
Distributed

- Every version of the repository, whether local or hosted (e.g., GitHub) is a full repository
- Transition to and from local copy to hosted repository, between hosted sites, and between local copies, etc.

Intro to Git

Git basics
A potential workflow

1. Say, we want to start/join a project, we start with an empty local directory/folder.
2. Every directory/folder could be a git ‘repository’ (repo)
3. Make a local copy of the project, i.e., clone it from the server
4. Start working on the project (making changes to the files)
5. Save each incremental progress
6. Perhaps, update the server.

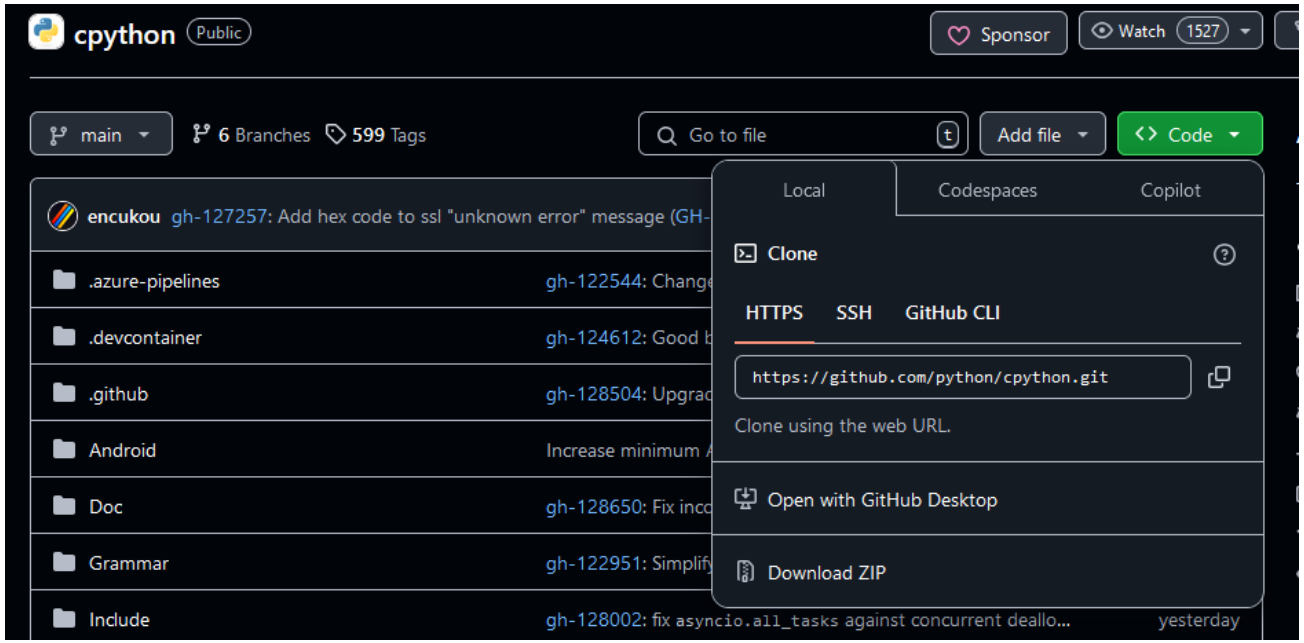


Git basics

A potential workflow

1. Say, we want to start/join a project, we start with an empty local directory/folder.
2. Every directory/folder could be a git ‘repository’ (repo)
3. Make a local copy of the project, i.e., clone it from the server
4. Start working on the project (making changes to the files)
5. Save each incremental progress
6. Perhaps, update the server.

1. Make/find a repo URL from GitHub



Git basics

A potential workflow

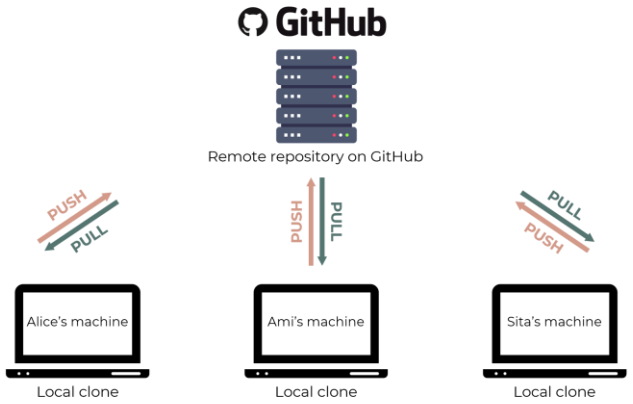
1. Say, we want to start/join a project, we start with an empty local directory/folder.
2. Every directory/folder could be a git ‘repository’ (repo)
3. Make a local copy of the project, i.e., clone it from the server
4. Start working on the project (making changes to the files)
5. Save each incremental progress
6. Perhaps, update the server.

1. Make/find a repo URL from GitHub https://github.com/python/cpython.git
2. Create an empty directory in your PC to save the repo.
3. Then clone it to a local directory to get a local copy of the remote repo.
4. Work on the code.
5. Then commit each change.
6. Finally, push the changes to the server.

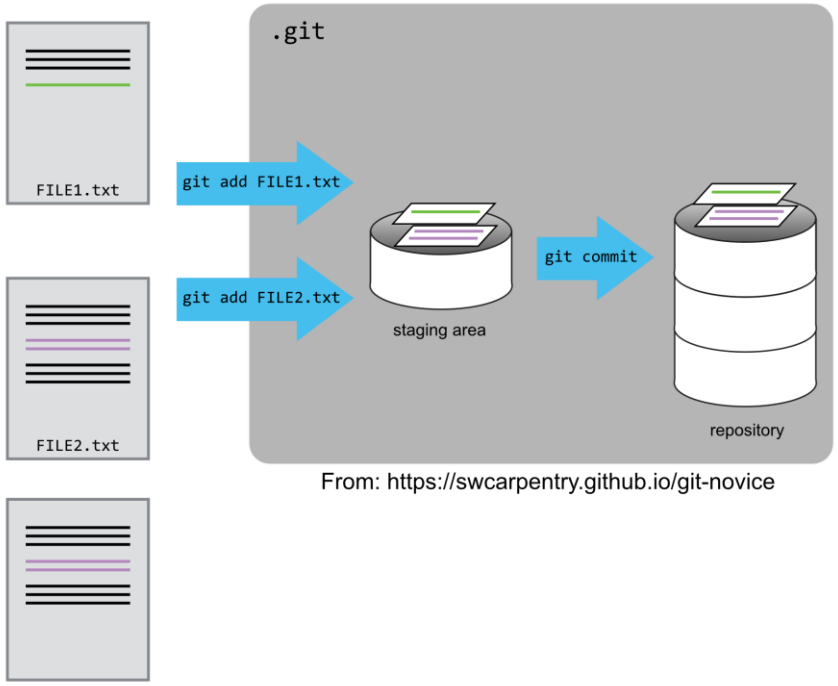
Simple git workflow

In the terminal

```
git clone https://github.com/python/cpython.git
cd cpython
git pull
… [make changes]
git status
git add . Or git add file.py
git status
git commit -m “add first test”
git push (if you have write access)
```



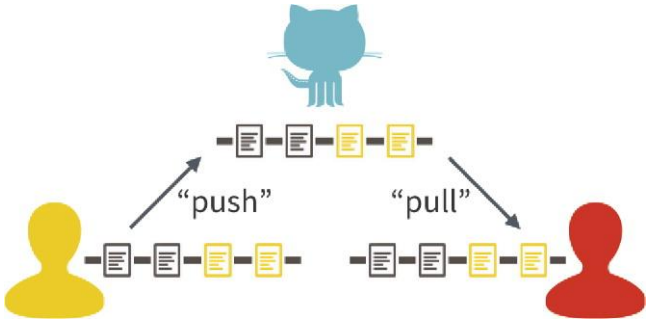
What if you don’t?



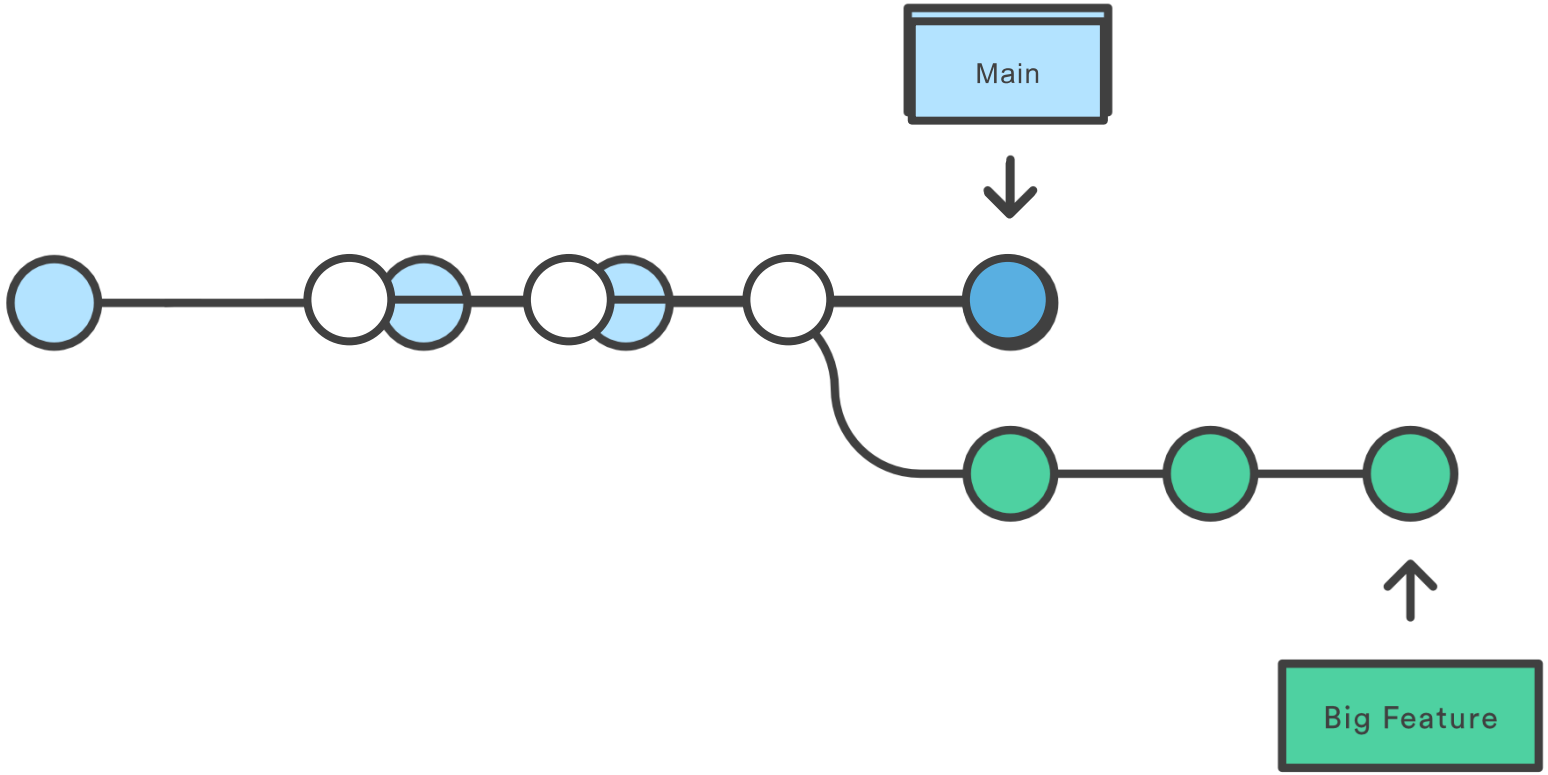
From: https://swcarpentry.github.io/git-novice

Branching
Basis of collaboration

```
git branch big-feature
git checkout big-feature
```
(add the feature via commits)
```
git push origin big-feature
```



https://css-skills.github.io/intro-to-git/

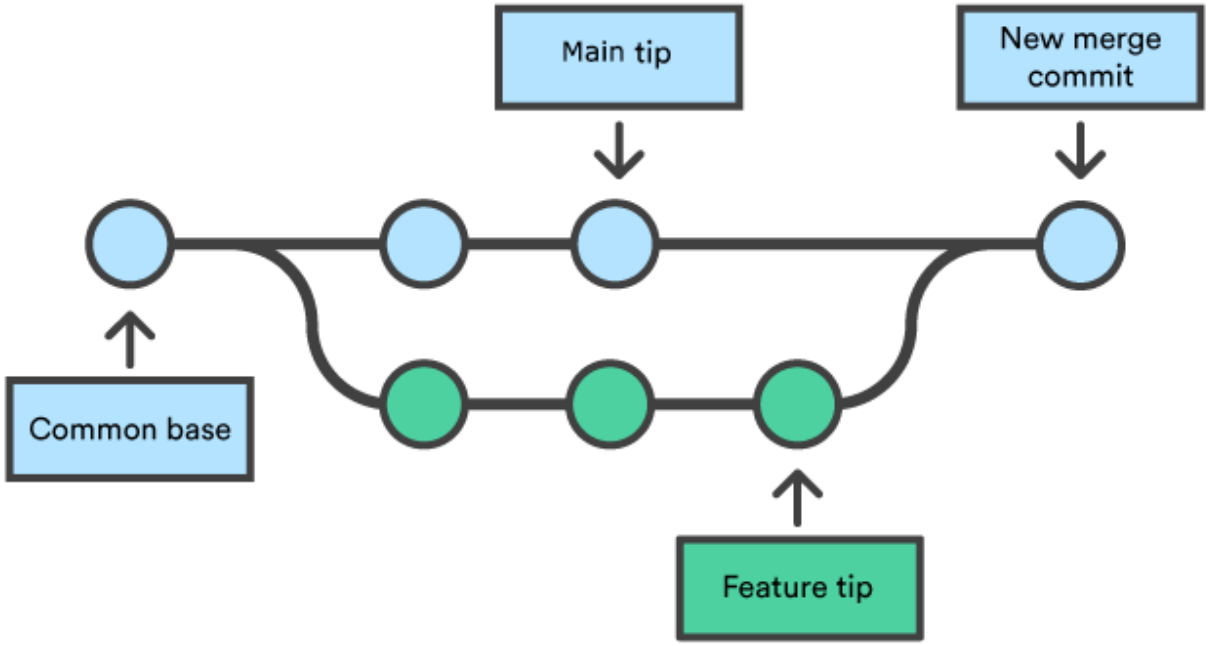


Branching

Merging the branch

```
git checkout main
```
(switch to the receiving branch)

```
git pull
git merge big-feature
git branch -d big-feature
```

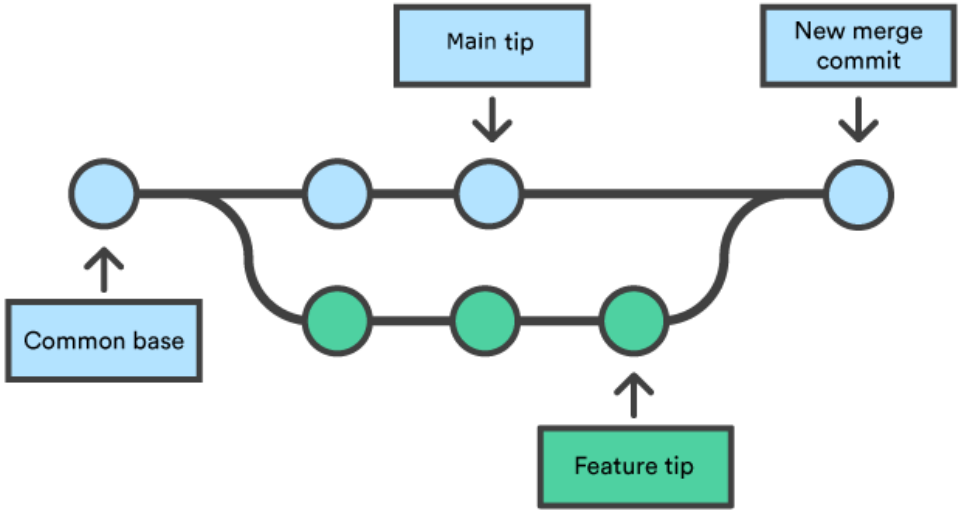


Merge conflicts

- F
- C
- C
- F
- C
- C

===, >>>>

```
$ git merge new_branch_to_merge_later
Auto-merging merge.txt
CONFLICT (content): Merge conflict in merge.txt
Automatic merge failed; fix conflicts and then commit the result.
```



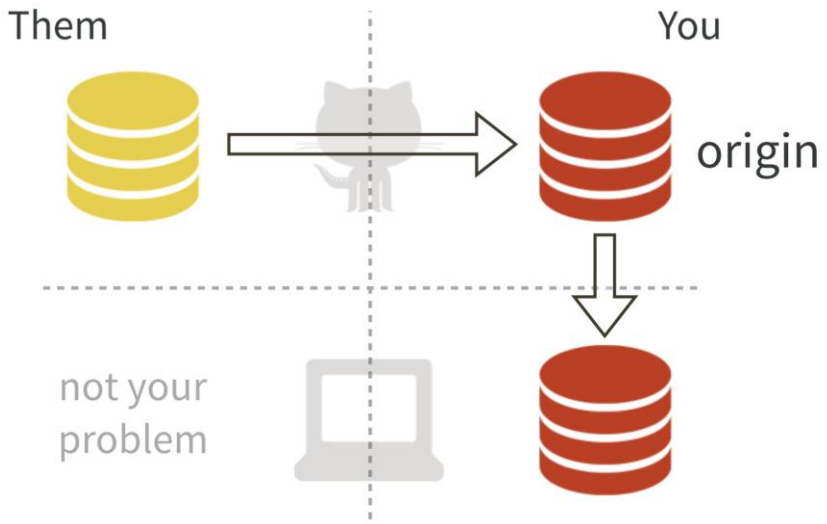

```
here is some content not affected by the conflict
<<<<<<< main
this is conflicted text from main
=======
this is conflicted text from feature branch
>>>>>>> feature branch;
```

Real world collaboration workflow

Forks and Pull Requests

How to contribute to an existing project?

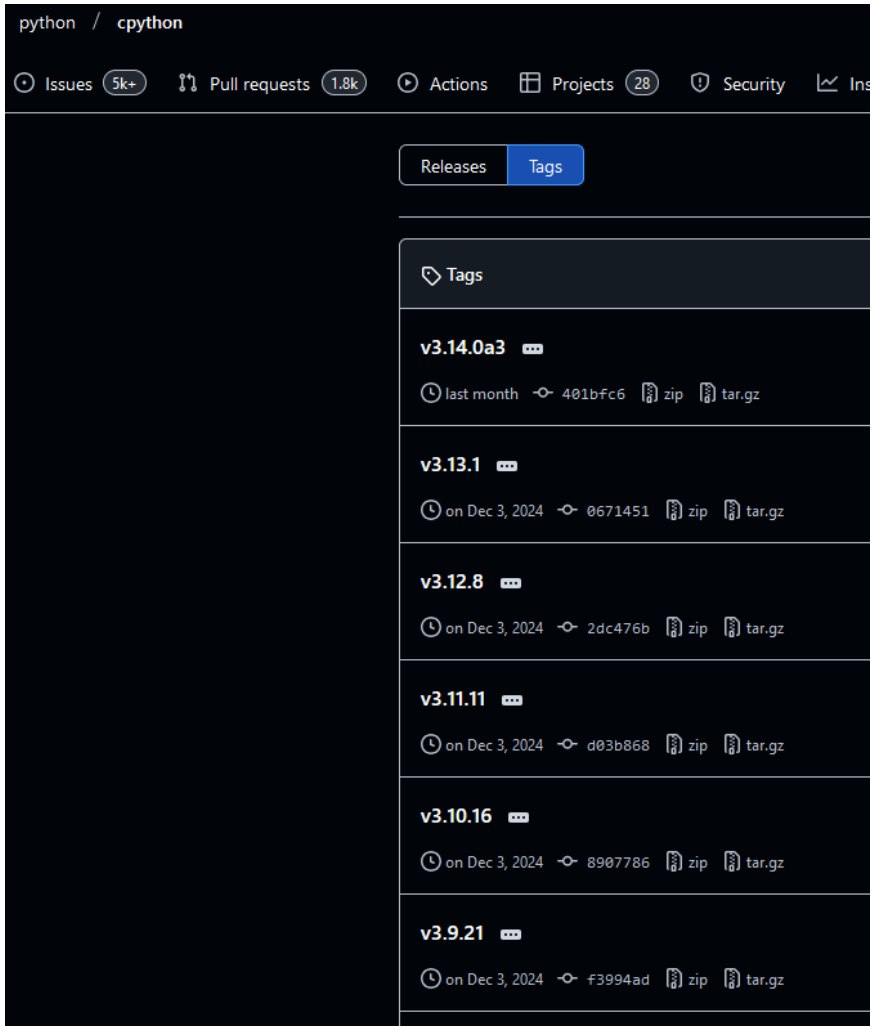
- Fork
  - Copy the repo in your personal GitHub
- Clone
- Make a branch for your work
- ...Code your idea in...
- Add and commit your work
- Check if it can merge with main branch
- Submit a Pull Request on GitHub
- Discuss with repo owner
- Owner might merge your PR



Tags

- References to specific points in the history of the repo
- Easier to access/find later
- Often used for version numbers of packages

- List stored tags: `git tag`
- Add a simple tag: `git tag v1.4`
- Add an annotated tag:
  `git tag -a v1.4 -m "my version 1.4"`



Ignoring files

Every file in the repo is either:

1. **Tracked**: previously committed
2. **Untracked**: A file which has not been committed, e.g., a newly added file
3. **Ignored**: a file explicitly ignored in `.gitignore`

The .gitignore content

- Use glob wildcards: `*.log`, `**/ __pycache__`
- Default GitHub ignored files: https://github.com/github/gitignore

PureScript.gitignore	Update PureScript adding .spago (#3278)	6 years ago
Python.gitignore	Merge branch 'main' into patch-1	last week
Qooxdoo.gitignore	Add gitignore for qooxdoo apps	15 years ago
Qt.gitignore	Remove trailing whitespace	4 years ago

gitignore / Python.gitignore

dooleydevin Merge branch 'main' into patch-1

Code Blame 174 lines (142 loc) · 3.36 KB

```
# Byte-compiled / optimized / DLL files
__pycache__/
*.py[cod]
*$py.class

# C extensions
*.so

# Distribution / packaging
.Python
build/
develop-eggs/
dist/
downloads/
eggs/
.eggs/
lib/
lib64/
parts/
sdist/
var/
wheels/
share/python-wheels/
*.egg-info/
.installed.cfg
*.egg
MANIFEST
```

A quick exercise

Basic git functions

Run the following commands

- Open Anaconda PowerShell (Windows) or a terminal
- `git config --global user.name “write your name”`
- `git config --global user.email type_your@email.address`
- `mkdir first_repo`
- `cd first_repo`
- `git init`
- `echo “import this” > test.py`
- `git status`
- `git add .`
- `git commit –m “add first test python script”`
- `git status`
- `git remote`

Want to see a little python Easter egg?

```
python test.py
```

Other useful git commands

Make note and read about them later ☺

- `git revert`
- `git reset`
- `git stash`
- `git rebase`
- `git fetch`
- `git cherry-pick`
- `git log`
- `git mv`

Good git practices

Commit every day, multiple times a day!

Got a question? Stackoverflow is your friend, ChatGPT may be too!

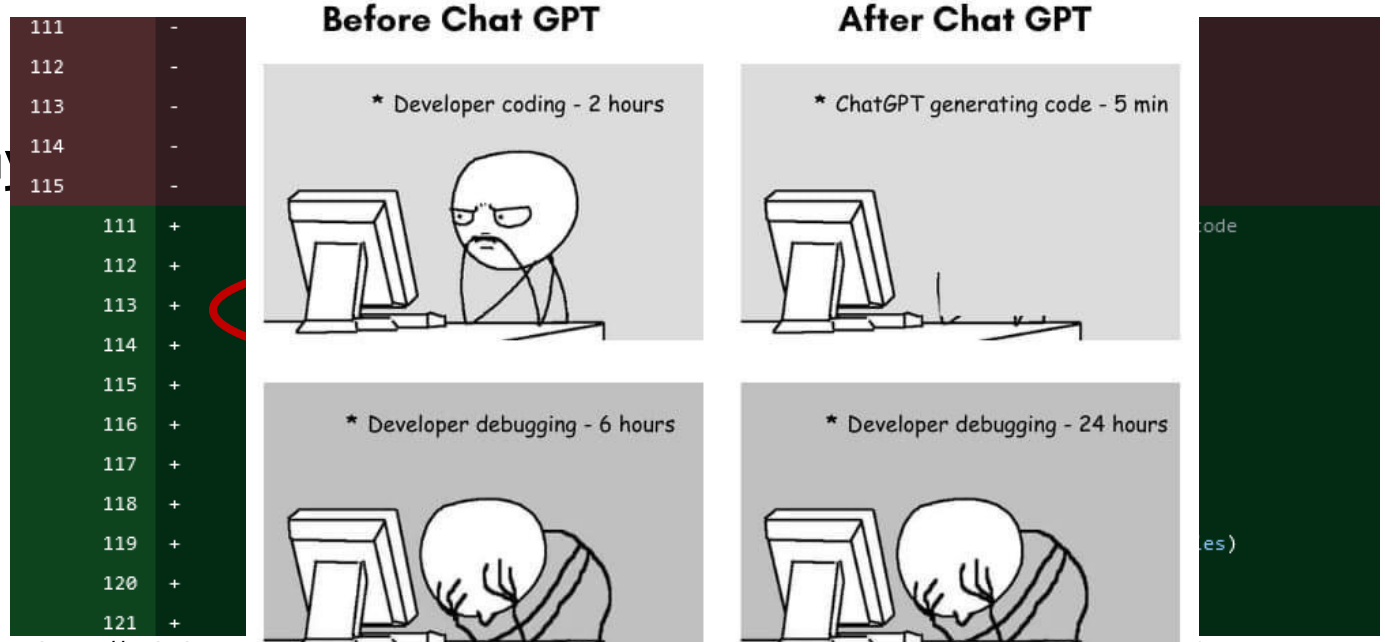
Check what a command does before running: `git rebase --help`

Beware of forcing a command:

- `--force`
- `-D` instead of `-d` in `git branch -D m`

Use graphical interfaces

Don't be scared—practice!



https://github.con ... 12a35074ce9

GitHub interface

Before moving to GitHub

A few tips and tricks

- Use a personal email for your account
  - You don’t want to lose access
- Link your University email to your account — Use GitHub Education
  - CoPilot for free
  - More private repos
- Set up SSH keys in your PCs to securely communicate with GitHub
- GitHub’s own graphical interface: *GitHub Desktop*
- In the URL of a repo, try changing `.com` to .dev

Now let’s open our browsers…

And go to github.com