

# Git Workshop



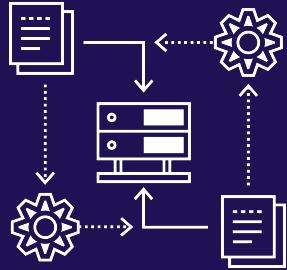
# Agenda

Hello 🙌

---

1. What's Git even? 🤔
2. Basics 🏫
3. Pull Request how-to 📜
4. Practice! 😱





# What's Git even?

# How does git work?

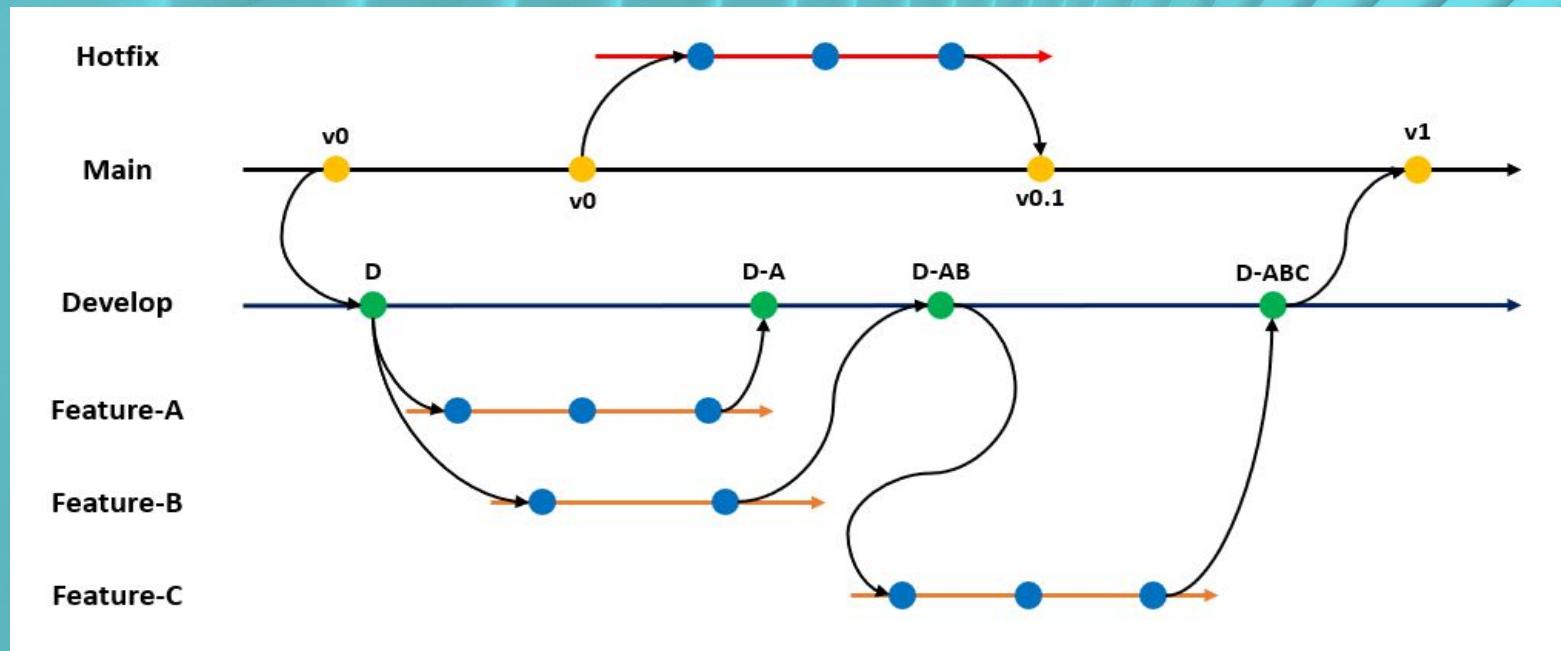
## Collaborations on trees

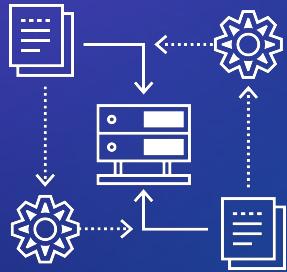
---

- has a tree-like structure
- every developer has a local copy on their machine
- you can change your local history

# Branches

```
git log --all --decorate --oneline --graph
```



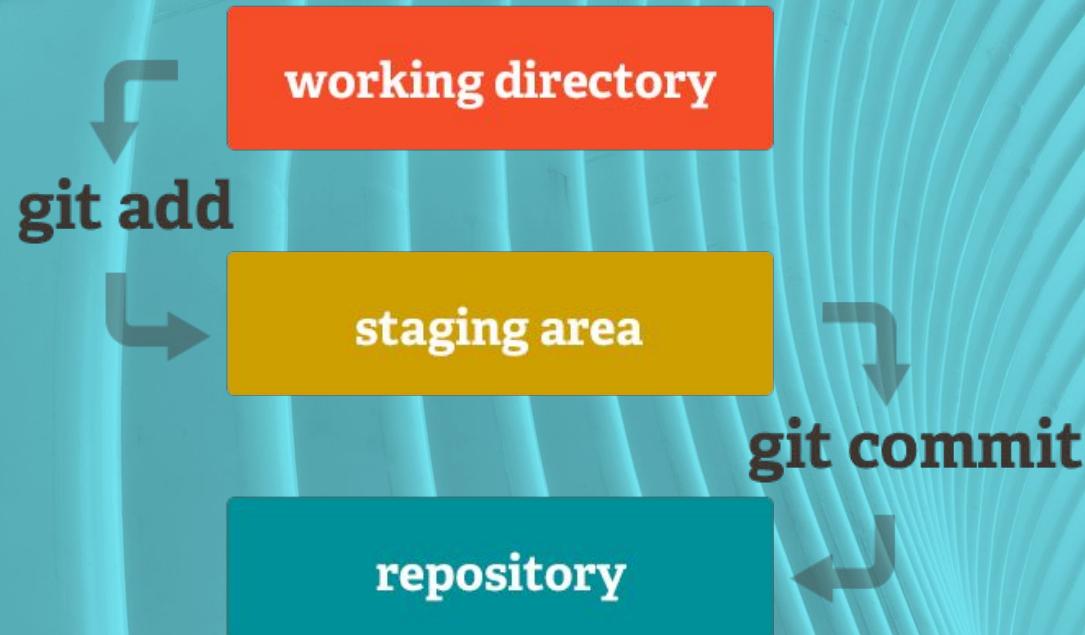


# Basics

# Getting started with git

What is working directory and staging area

---



# Basic commands

**add**

```
git add [file/directory]
```

Stage files for commit.

**Good practice:**  
Don't use *git add .*

**status**

```
git status
```

Show current list of files with changes.

**commit**

```
git commit -m [message]
```

Commit changes with message.

**checkout**

```
git checkout [branch]
```

Add -b to create a new branch.

Add -d to delete a branch.

# Push & Pull

**Me when  
the door  
says **pull**:**



**Me when  
the door  
says **push**:**



**push**

git push [from] [to]

Push changes to a  
branch.



**pull**

git pull [to] [from]

Pull changes from a  
branch.

# Useful commands

**stash**

git stash show - show latest stash  
git stash list - show list of stashes  
git stash - save changes into stash (removes from workspace!)  
git stash apply/pop - apply stashed changes (pop removes from stash list)

**fetch**

git fetch [to][branch] - saves a remote work in your local repo (without making actual changes, so you have to git checkout)

**restore**

git restore [file] - reverse the changes made on a file

# Process summary

Clone the repository with:

```
git clone <repository>
```

No need to add to remote  
list

**Always pull changes from  
dev before working!**

*Create a new branch*  
git checkout -b <name>

*~do stuff~*

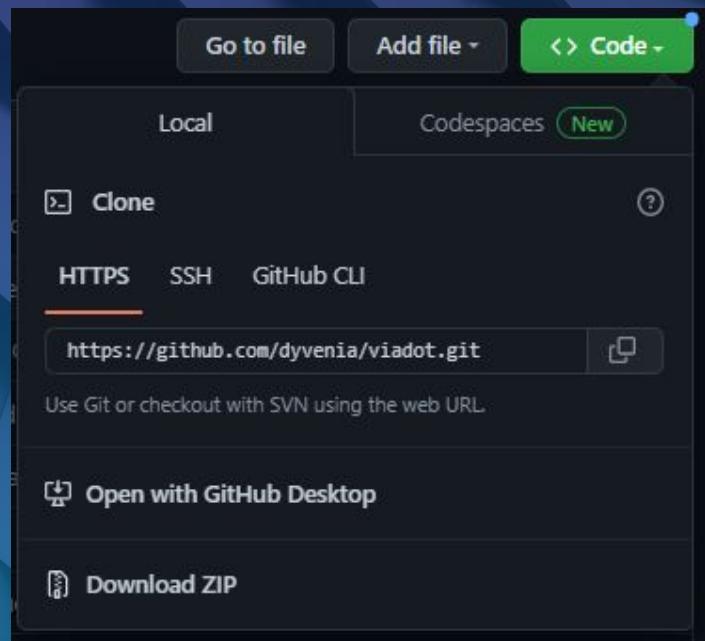
Add and commit  
git add <things>  
git commit -m "message"

*Push changes and merge*  
git push <origin> <branch>

# Connecting to the repository HTTPS

## Scheme for downloading the repository

1. Next to the repository you want to download there is a green "Code" button, click this button
2. Copy HTTPS URL
3. On your local computer, go to the directory where you want to download the repository.
4. Use the command git clone <URL>



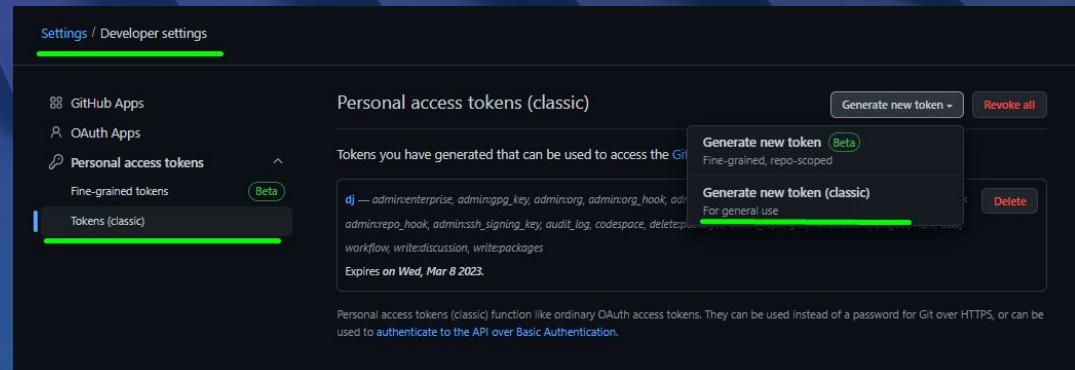
# Git token generation

Every time you want to "push" or "pull" you will be asked by git for a Personal access token

---

Token generation:

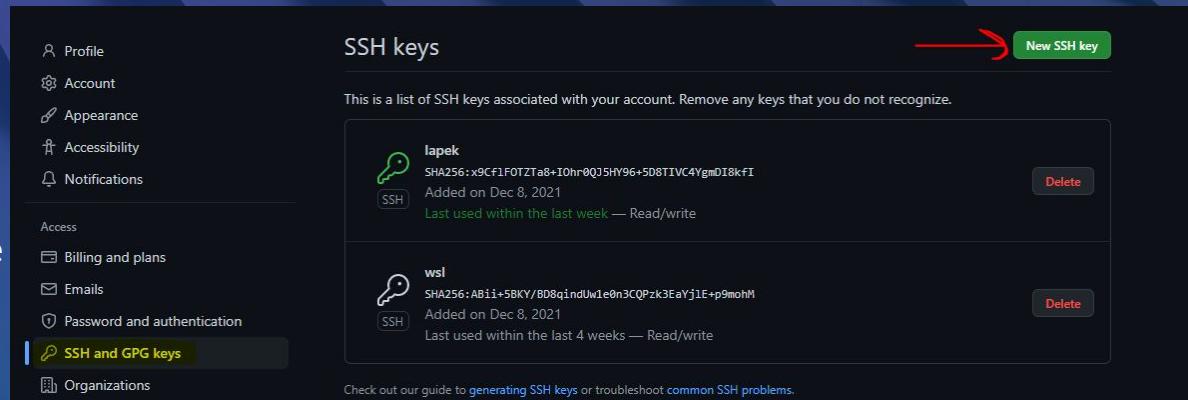
1. Go to account settings
2. Select Developer settings
3. Choose Personal Access Tokens
4. Generate token



# Connecting to the repository SSH

**ssh-keygen -t rsa -b 4096 -C "tag-of-your-ssh"**

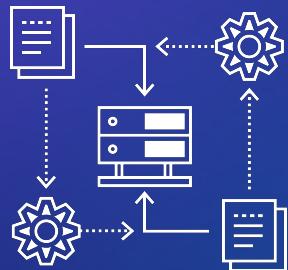
1. Create an SSH key
2. Save it to a file
3. Write passphrase
4. ssh-add -K /path/to/your/file
5. Add SSH on git



The screenshot shows the GitHub 'SSH keys' page. On the left, there's a sidebar with options like Profile, Account, Appearance, Accessibility, Notifications, Access, Billing and plans, Emails, Password and authentication, and **SSH and GPG keys** (which is highlighted). Below the sidebar is an 'Organizations' section. The main area is titled 'SSH keys' and contains two entries:

- lapek**  
SHA256: x9CF1FOTZTa8+1Ohr0QJ5HY96+5D8T1VC4YgmDI8kfI  
Added on Dec 8, 2021  
Last used within the last week — Read/write  
[Delete](#)
- wsl**  
SHA256: Abii+5KY/BD8qindUw1e0n3CPzk3EaYj1E+p9mohM  
Added on Dec 8, 2021  
Last used within the last 4 weeks — Read/write  
[Delete](#)

At the bottom of the main area, there's a link to a guide: [generating SSH keys](#) or [troubleshoot common SSH problems](#).



# Fork

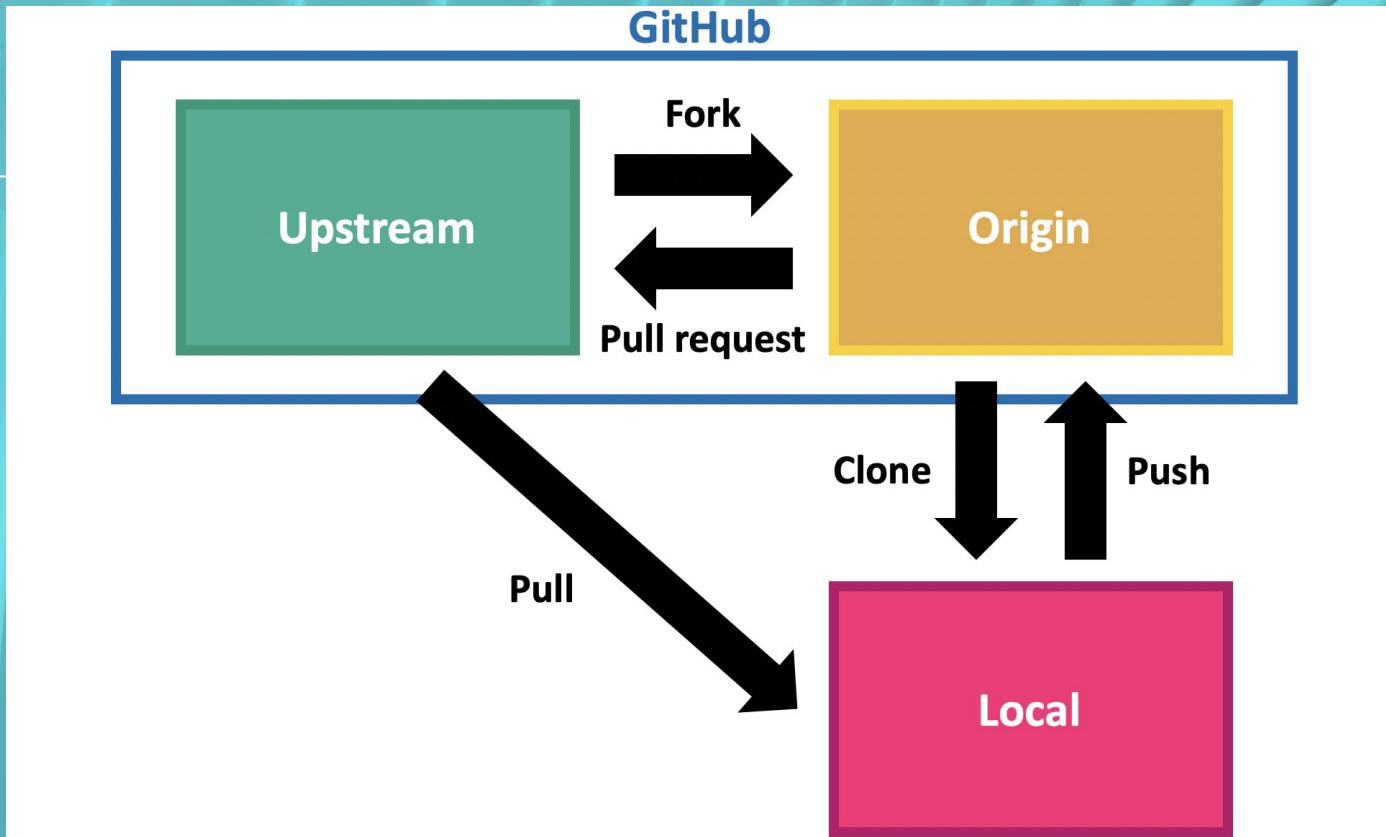
# About forks

---

A fork is a copy of a repository that you manage. Forks let you make changes to a project without affecting the original repository. You can fetch updates from or submit changes to the original repository with pull requests. Forking a repository is similar to copying a repository, with two major differences:

- You can use a pull request to suggest changes from your user-owned fork to the original repository in its GitHub instance, also known as the upstream repository.
- You can bring changes from the upstream repository to your local fork by synchronizing your fork with the upstream repository.

# Fork workflow



# Forking

On github repository page,  
click ‘fork’. Clone the forked  
repository with:

```
git clone <repository>
```

Add the original repository to  
your remote list

```
git remote add <name> <url>
```

**Always pull changes from  
dev before working!**

*Create a new branch*

```
git checkout -b <name>
```

*~do stuff~*

Add and commit

```
git add <things>
```

```
git commit -m "message"
```

*Push changes and merge*

```
git push origin <name>
```

# Conflicts

# Basic commands

**rebase**`git rebase [branch]`

Update branch based on changes made.

Switches your branch's base to the other branch's position and walks through your commits one by one to apply them again.

**merge**`git merge [branch]`

Update branch based on changes made.

Creating a new commit that incorporates the changes of both branches.

**checkout**`git checkout [branch]`

Change branch.

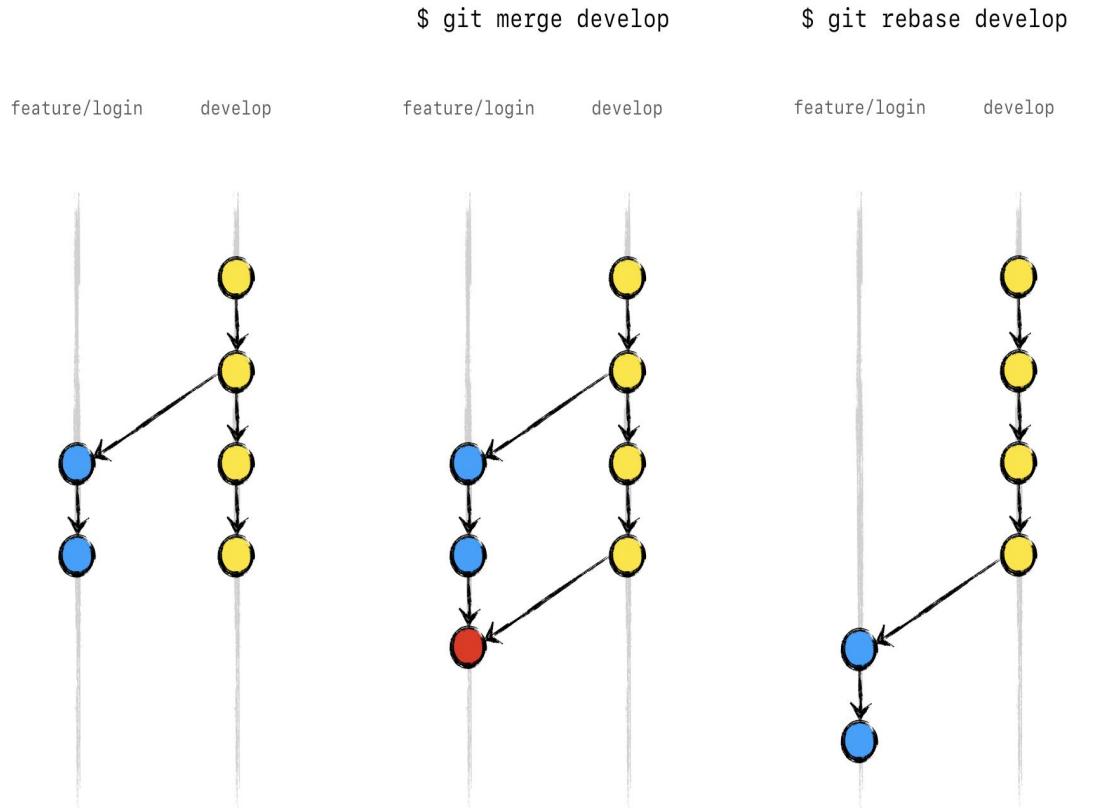
Add -b to create a new branch.

Add -d to delete a branch.

**status**`git status`

Show current list of files with changes.

# Difference dyvenia between rebase and merge



Git's rebase command reapplies your changes onto another branch. As opposed to merging, which pulls the differences from the other branch into yours, rebasing switches your branch's base to the other branch's position and walks through your commits one by one to apply them again.

**Remember to do a rebase if changes have been made to the main/dev branch**

When you are gonna merge two branches after long time..

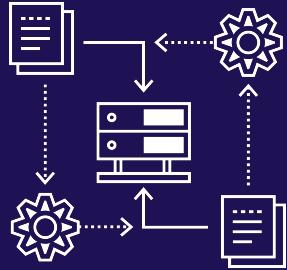


# Merge conflicts

Open your VSCode

---

```
        '
You, 3 minutes ago | 2 authors (You and others) | Accept Current Change | Accept Incoming Change | Accept Both Changes | C
<<<<< HEAD (Current Change)
print("Welcome to the git program!")
=====
print("Welcome to the workshop program!")
>>>>> 5fa8e122651f8755ea4cb0fb18bcfe43c1fe3c05 (Incoming Change)
print("I'm not very useful")
```



# Pull Request how-to

# Keep it simple

## **One PR should implement one functionality**

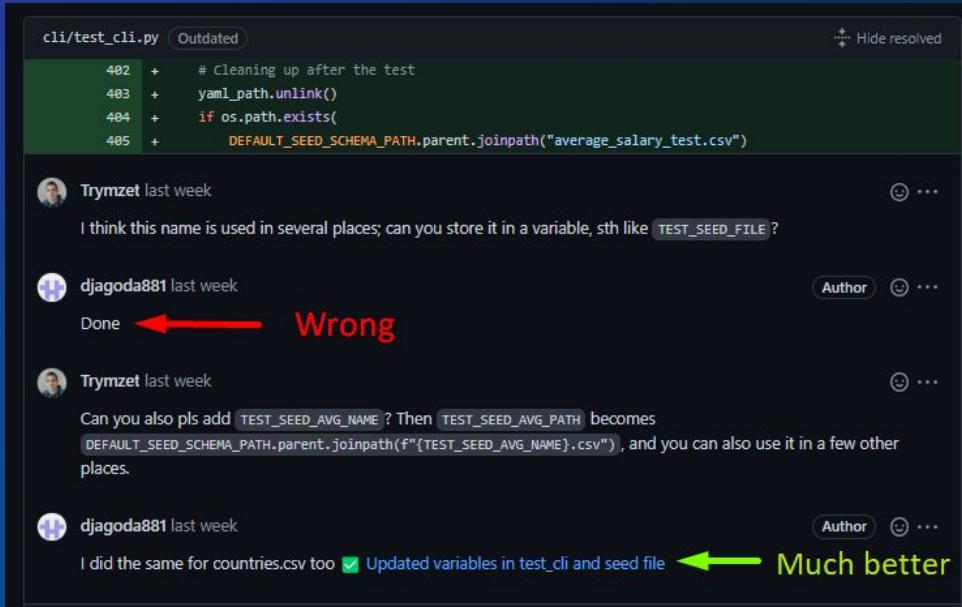
---

**Don't try to push as much as possible** - if you notice a bug, have an idea for a new component get a sudden urge to write a test for something not directly connected to what you're working on - leave it for next PR. Open an issue about it.

**Keep the amount of commits per PR as low as possible.** Reading through 100 commits is not the most pleasurable thing to do. But so is reading one commit with 20.000 lines of code. It's all about the balance in the universe.

# Code Review

It relies on the fact that the code we write, before it goes to the main branch, is reviewed by a second developer called a reviewer in the process.



The screenshot shows a GitHub pull request interface for a file named `cli/test_cli.py`. The code has been updated, as indicated by the "Outdated" status. The changes are:

```
402 +     # Cleaning up after the test
403 +     yaml_path.unlink()
404 +     if os.path.exists(
405 +         DEFAULT_SEED_SCHEMA_PATH.parent.joinpath("average_salary_test.csv")
```

A comment from `Trymzet` last week suggests storing the path in a variable:

I think this name is used in several places; can you store it in a variable, sth like `TEST_SEED_FILE` ?

Another comment from `djagoda881` last week, with a red arrow pointing to the word "Wrong", indicates that this suggestion was incorrect:

Done ← Wrong

Further comments from `Trymzet` suggest adding a new variable `TEST_SEED_AVG_NAME`:

Can you also pls add `TEST_SEED_AVG_NAME` ? Then `TEST_SEED_AVG_PATH` becomes  
`DEFAULT_SEED_SCHEMA_PATH.parent.joinpath(f"{TEST_SEED_AVG_NAME}.csv")`, and you can also use it in a few other places.

Finally, a comment from `djagoda881` last week, with a green arrow pointing to the word "Much better", indicates that they have implemented the suggested changes:

I did the same for countries.csv too ✓ Updated variables in test\_cli and seed file ← Much better

# Naming conventions

## :emoji: Verb + functionality

---

Always begin with a gitemoji (VSCode extension, each emoji specifies a function).

Use verbs such as “Added” or “Fixed”, capitalized.

Specify the functionality, try to keep it short.

If you get lost - naming conventions are usually written down in the documentation.

# Examples

## Bad

---

- Fixed a bug
- 🍴 Added a test for \*my-thing\*
- anna\_commit
- ✨ Added the possibility to parse a letter from a non-latin alphabet - a mandarin Chinese symbol for 'data', I've discovered those by reading a book once mentioned from my father whom...

## Better

---

- 🐛 Fixed a bug in \*name\* for empty df
- ✓ Added a test for my thing
- 🎨 Updated code structure for \*name\*
- ✨ Added the possibility to parse a letter from a non-latin alphabet

# PRACTICE TIME



YOU MADE  
YOUR  
FIRST PUSH



THE  
STRUCTURE LOOKS  
A BIT DIFFERENT



YOU WIPE  
OUT THE  
WHOLE REPO

# **Thanks!**

Contact us:

ul. Czyżówka 14/0.3,  
30-526 Kraków  
Poland

[hello@dyvenia.com](mailto:hello@dyvenia.com)  
[www.dyvenia.com](http://www.dyvenia.com)

