



## Git - Part I

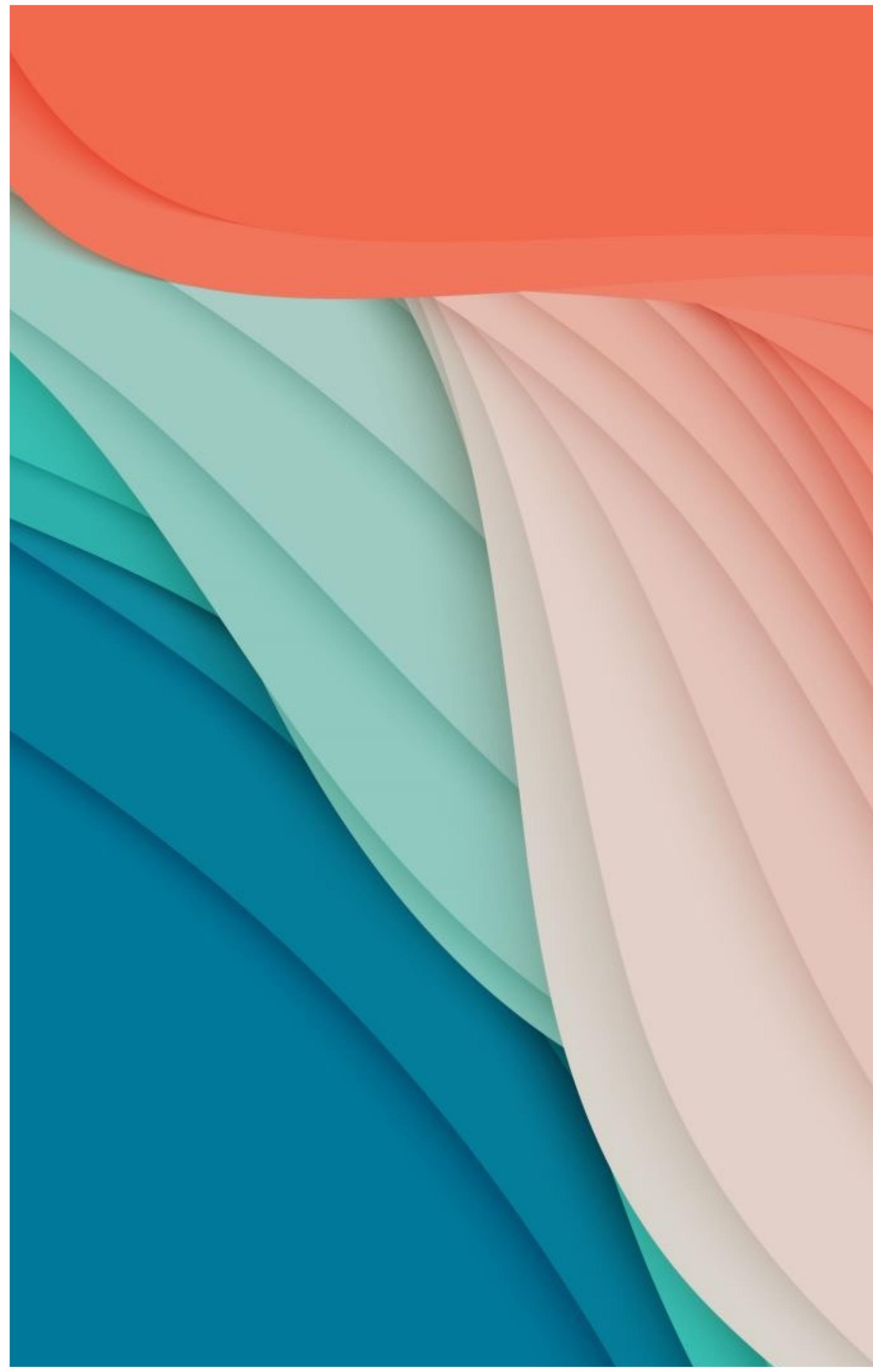
# Introduction & Basics

---

Amine Ferdjaoui

[amine.ferdjaoui@sogeti.com](mailto:amine.ferdjaoui@sogeti.com)

Année universitaire 2025/2026



# Contenu du cours

---

1. Initiation à git : commandes de base.
2. Utilisation de Visual Studio Code.
3. Branches dans git.
4. Collaboration dans git (GitHub).
5. Bases de docker.
6. Utilisation de docker hub.
7. Docker compose.



Projet



Projet final



Projet final 2



Projet final 3



Projet final final



Projet final finaaaal



finaaaaaaaaaaaal

# What is Git ?

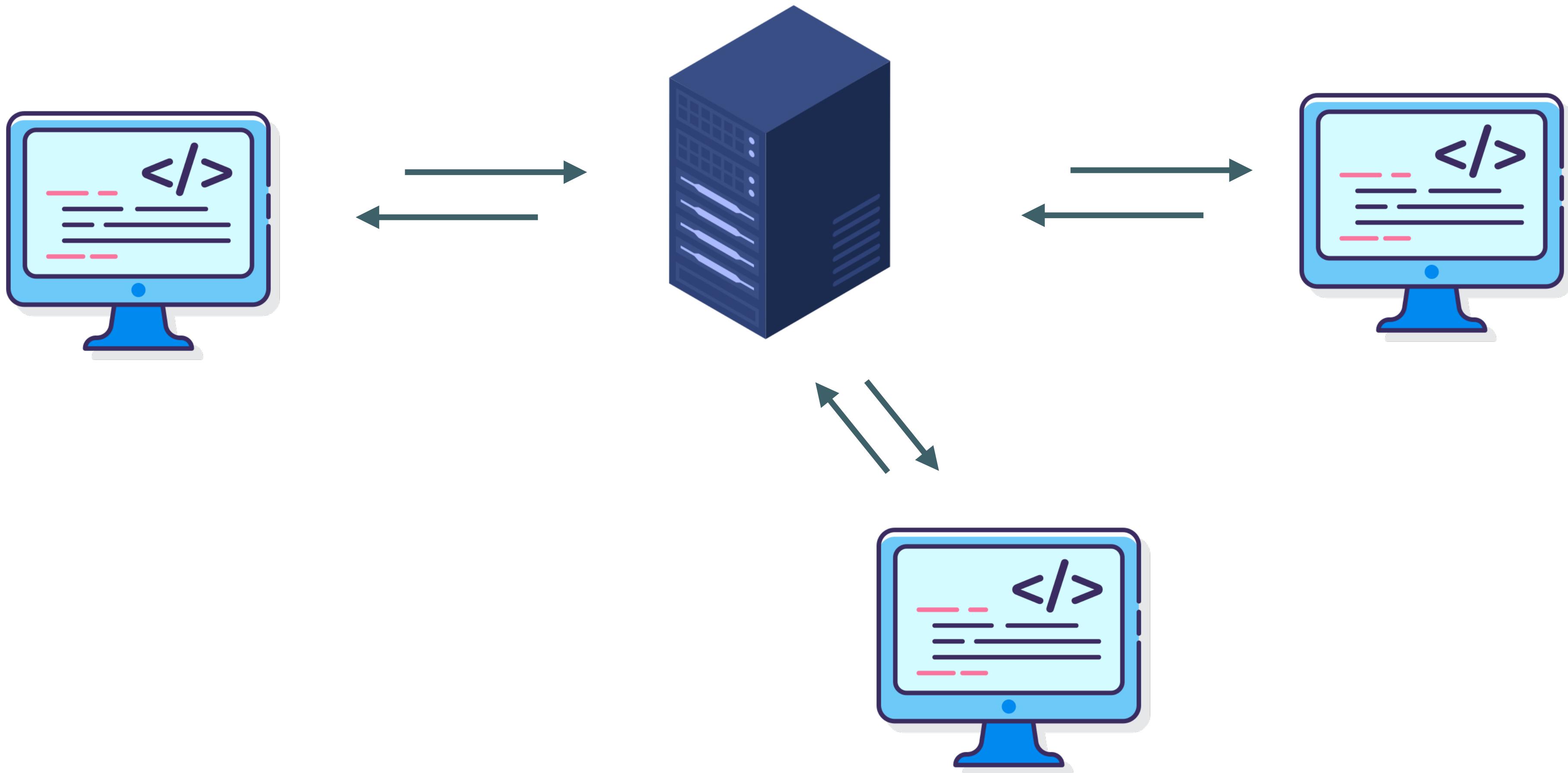
---

- ◆ Git is a **distributed** version control system (VCS).
- ◆ Track changes made on files and revert back to them (like a time traveling machine).
- ◆ Developed and released in 2005 by Linus Torvalds, the creator of Linux.
- ◆ It is a free and open source.
- ◆ According to a [Stack Overflow developer survey](#) over 87% of developers use Git.

# What is a distributed VCS

---

*No single point of failure*



# Git hosting service

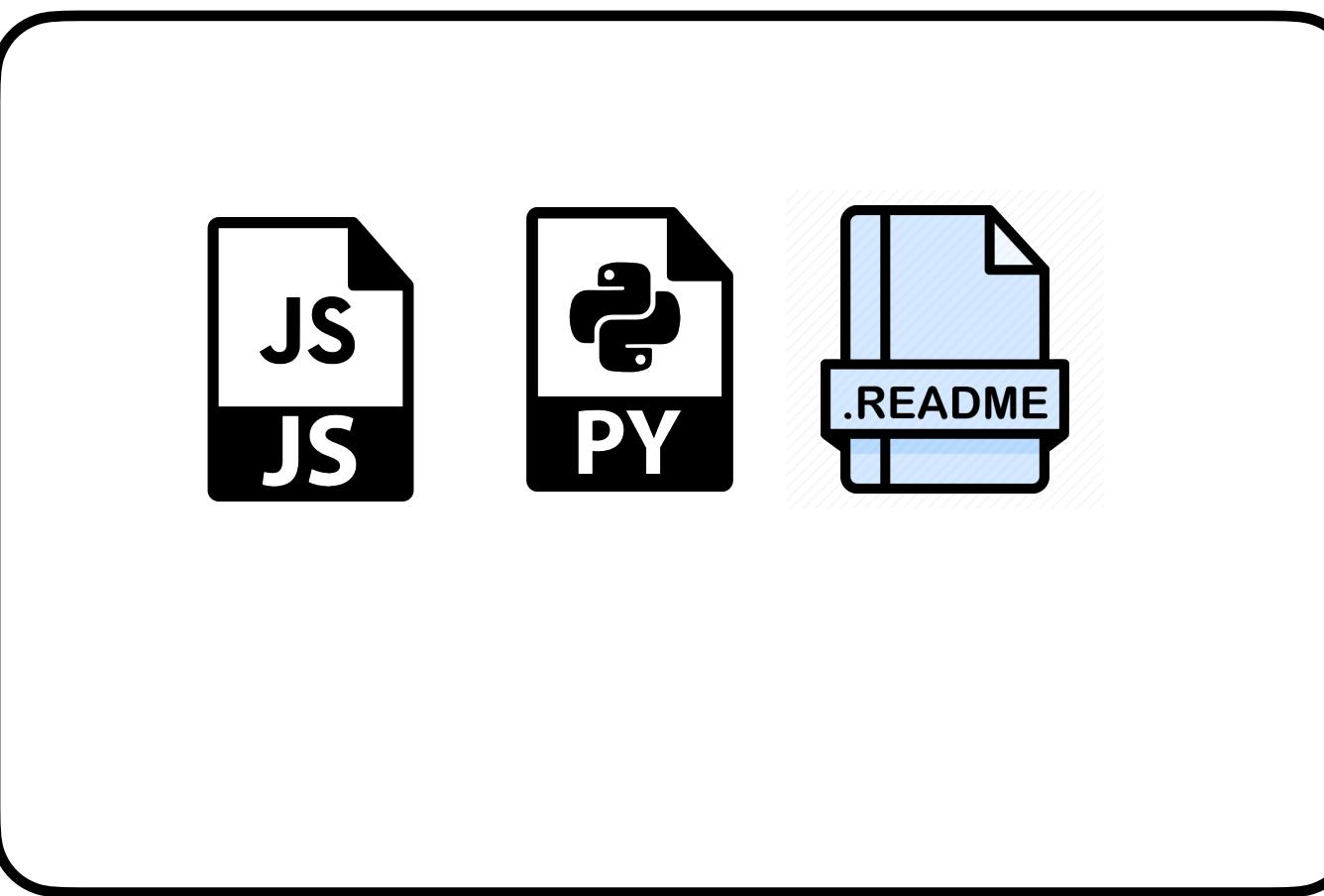
---

- ◆ Web hosting service that hosts git projects on the cloud.
- ◆ The most used ones are : Github, bitbucket and gitlab.
- ◆ **Github** is the most popular one, it was bought by Microsoft in 2018.

# General git workflow

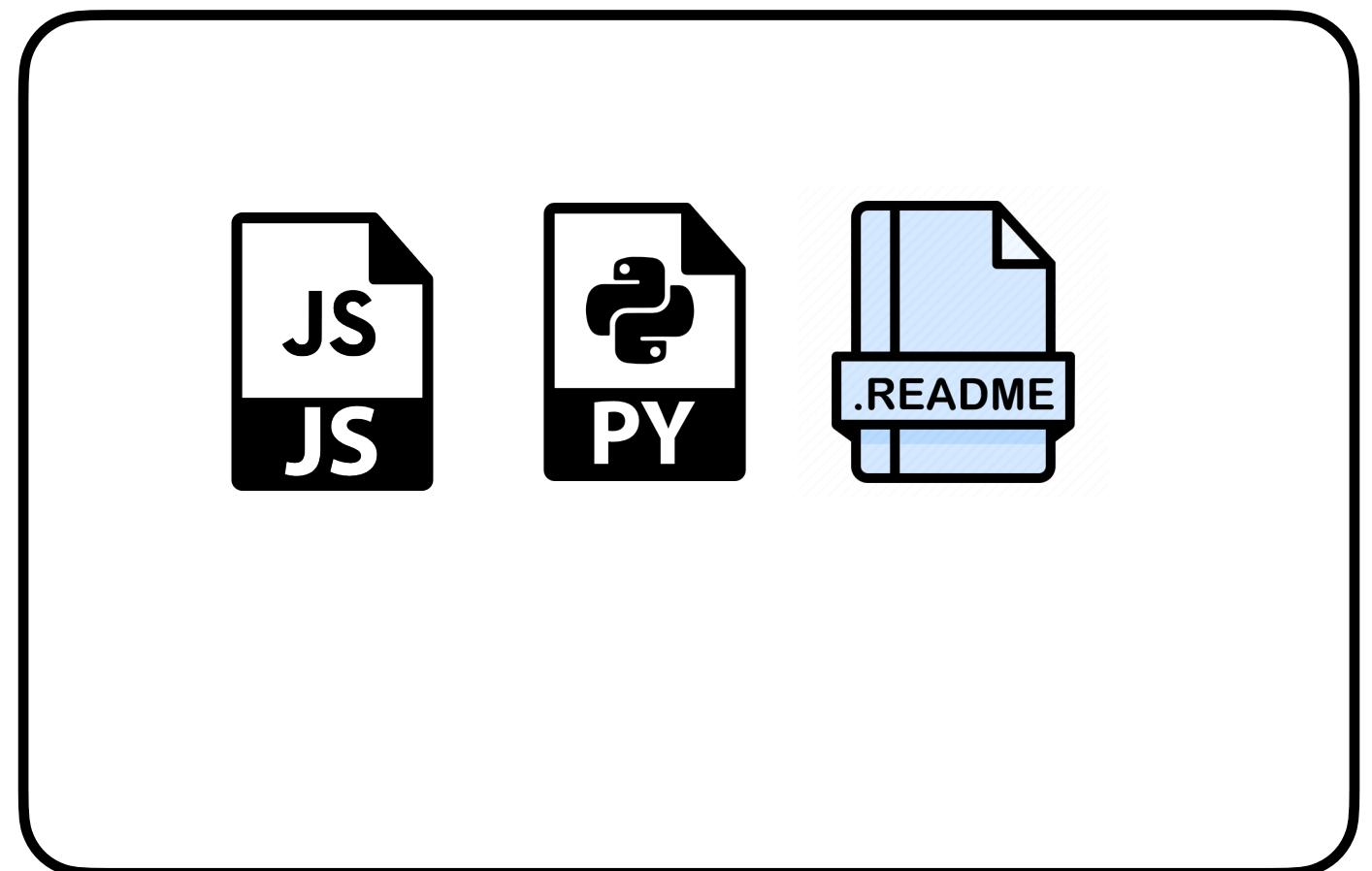
---

## Working directory



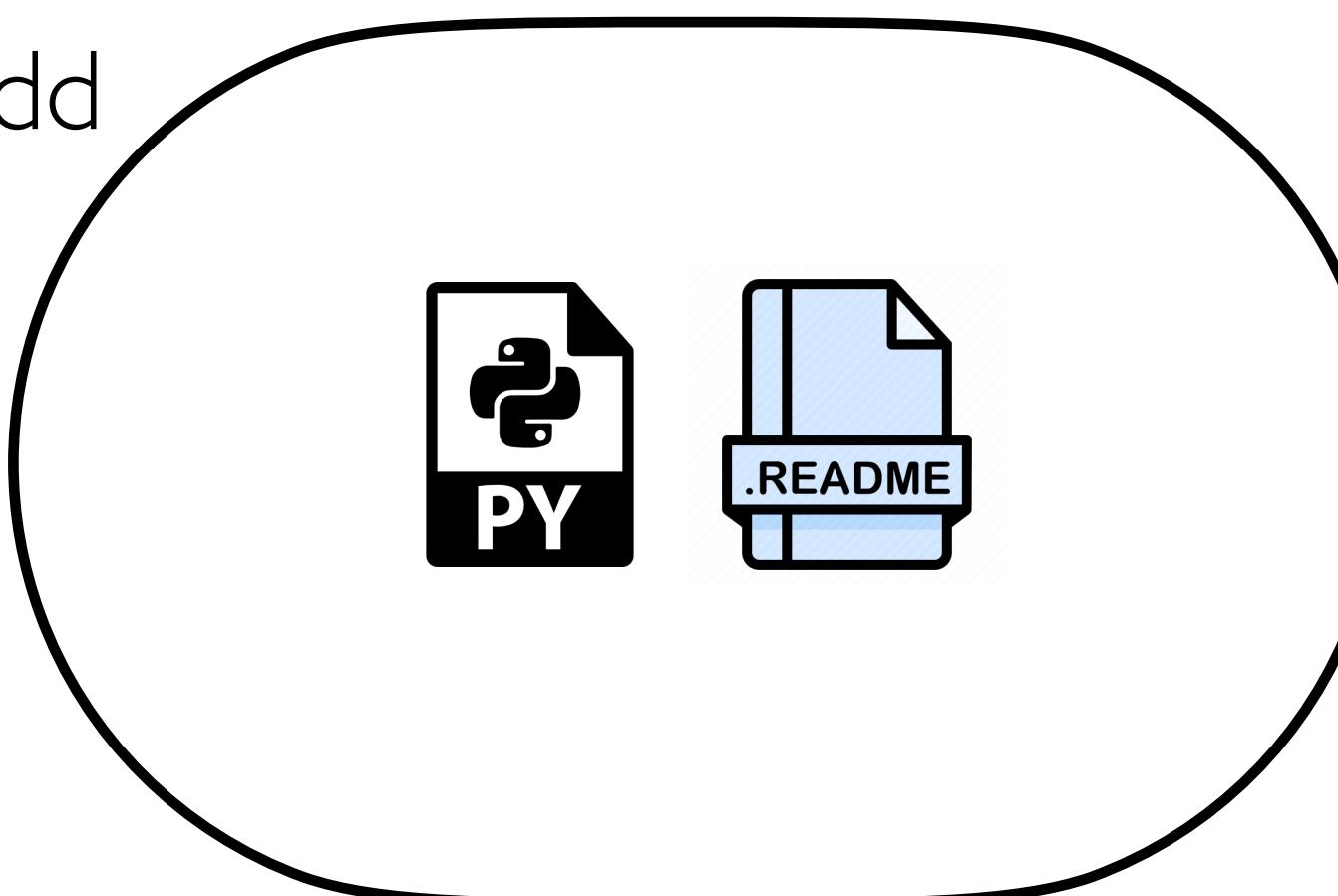
# General git workflow

## Working directory



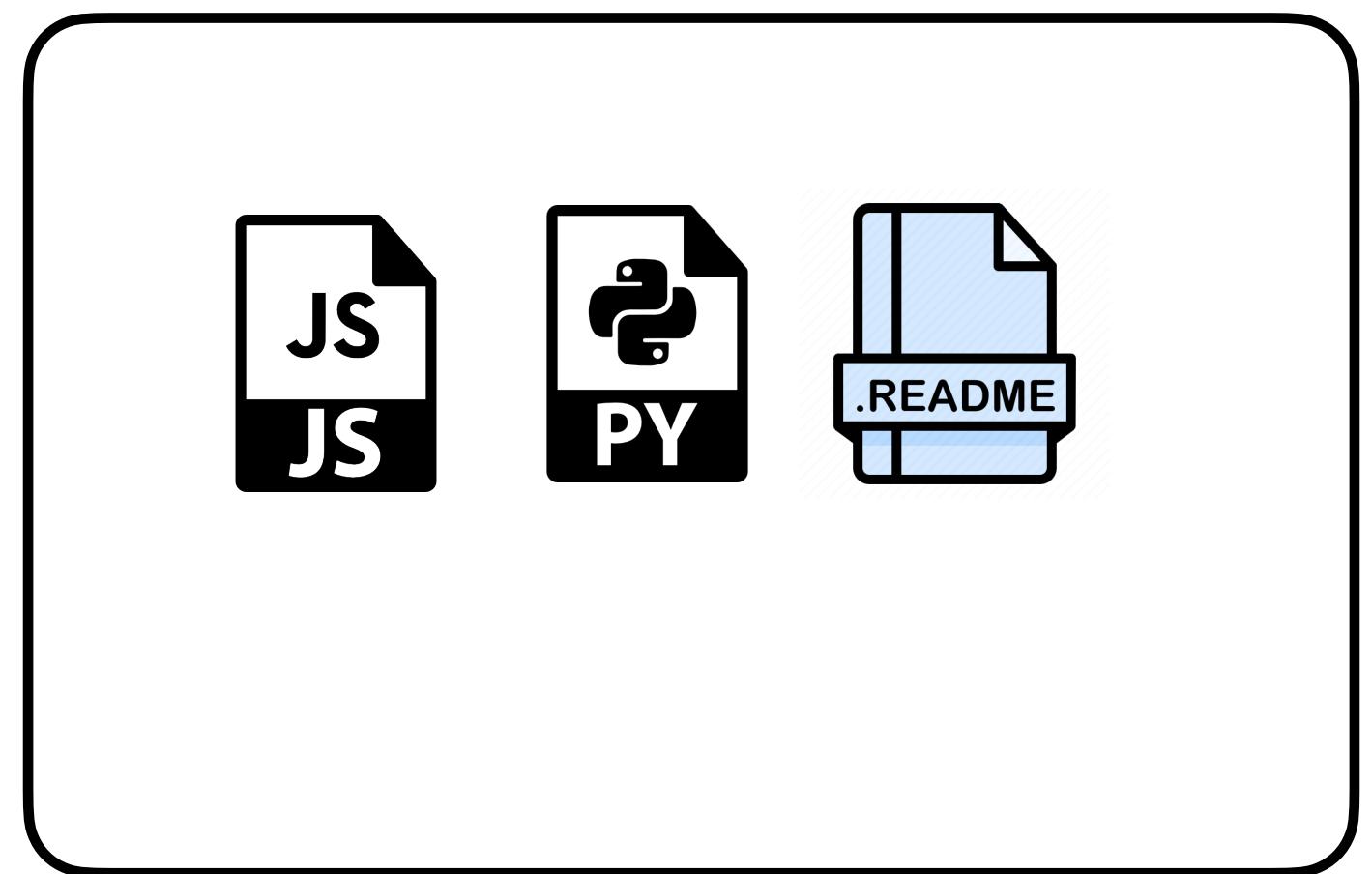
**Staging**  
or Index (old name)

Git add



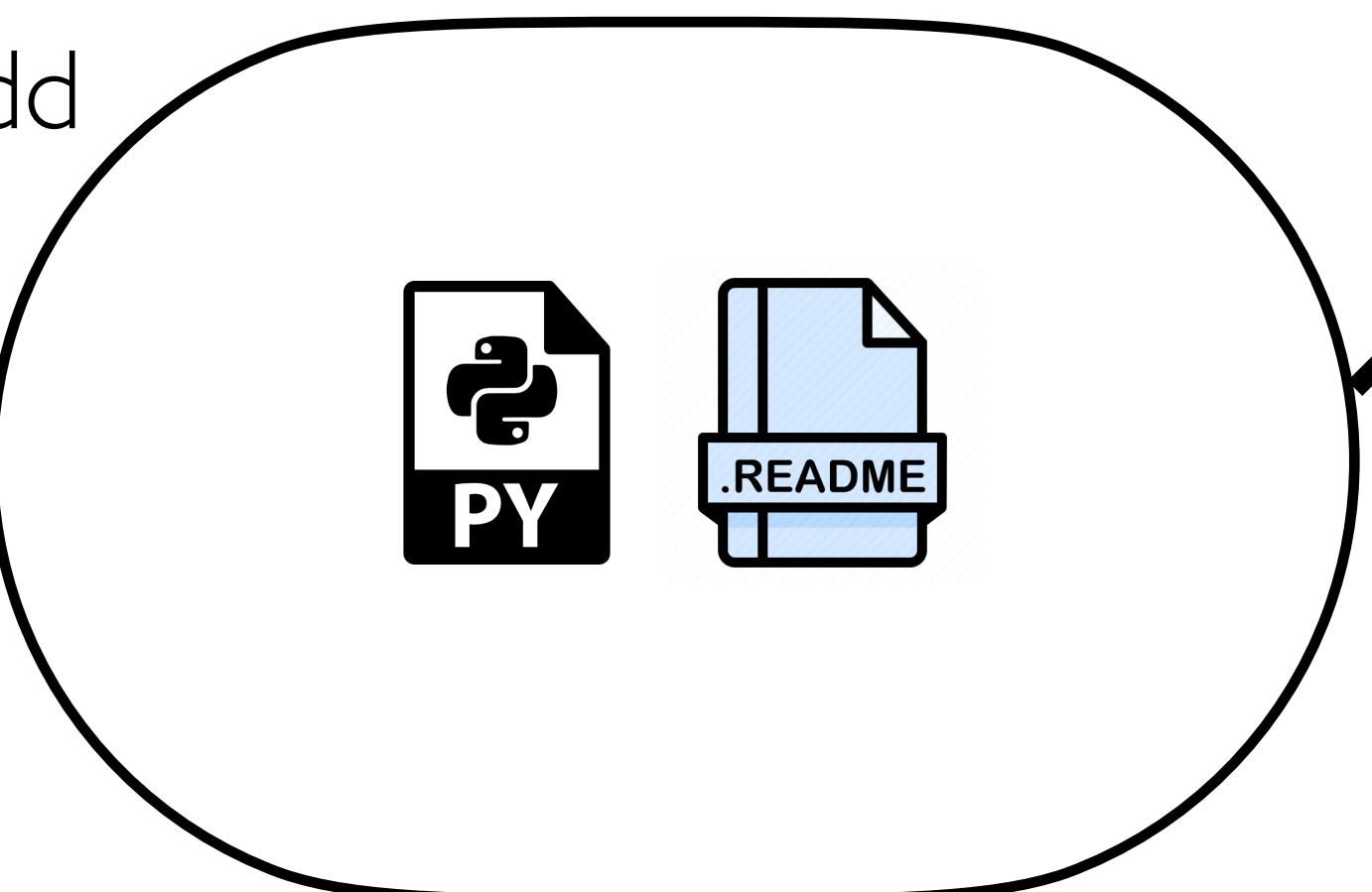
# General git workflow

## Working directory

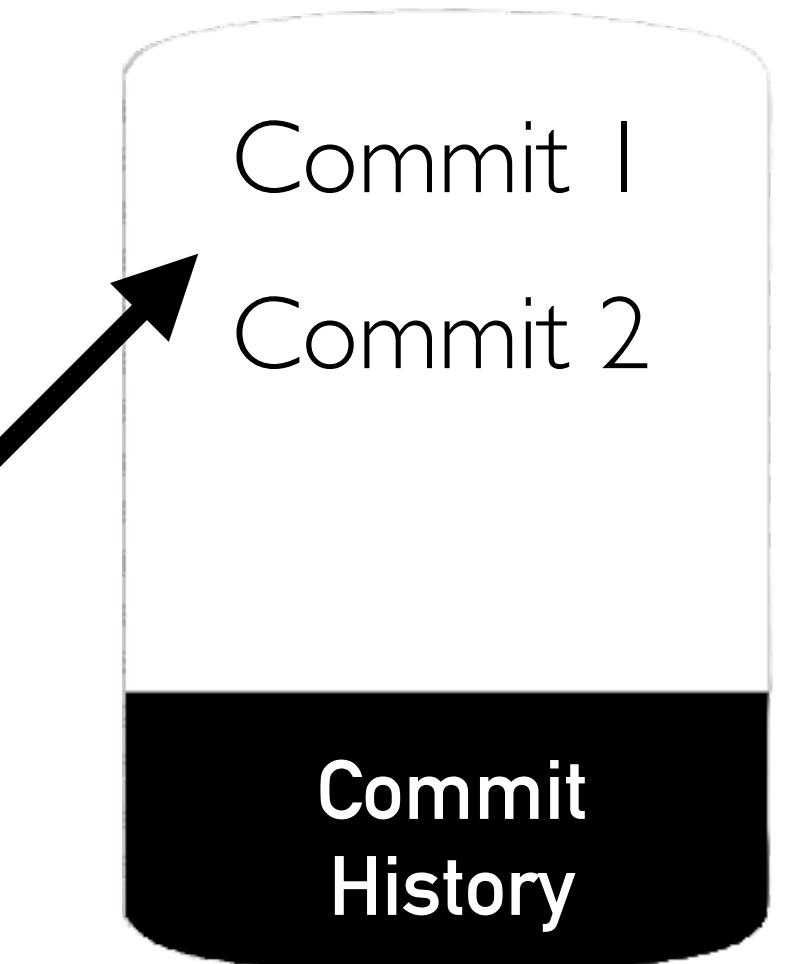


## Staging or Index (old name)

Git add

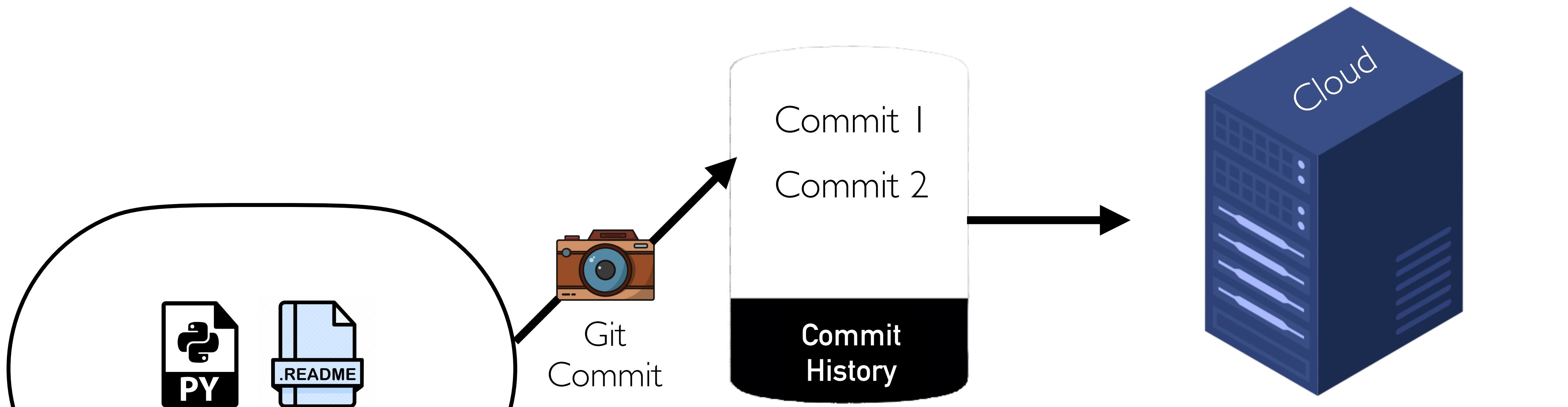


Git Commit



Ps. Git doesn't save the same files for each commit.  
Files are compressed and saved uniquely

# General git workflow



**Staging / Index (old name)**

**Hosting service**

Github

Bitbucket

Gitlab

etc.

# Git installation

---

- ◆ Installation instructions : <https://github.com/git-guides/install-git>
- ◆ For windows use Git BASH

```
$ git version  
git version ***
```

# Git configuration

Cf. <https://git-scm.com/book/sv/v2/Customizing-Git-Git-Configuration>

```
$ git config [config level] [config name] [config value]
```

- ◆ [config levels] :

- local : (default) applies to the local repository, config is found in .git/config

- global : applies to the OS user repositories, config is found in ~/.gitconfig

- system : applies to the all OS users, config is found in /etc/gitconfig

- ◆ [config Name] :

```
$ git config --global user.name 'amine'
```

- user : email | name

```
$ git config --global user.email 'youremail@domain.com'
```

- core : editor

- alias : git command

```
$ git config --local alias.st status -s
```

```
$ git config --global alias.lg 'log --oneline --graph --all'
```

```
$ git config --global -e #To see the current configuration
```

# Bash commandes reminder

\$ cd

\$ ls -A

\$ pwd

\$ chmod

\$ grep

\$ mv

\$ cp

\$ rm -r

\$ mkdir

\$ touch

\$ cat

\$ diff

\$ echo

\$ man

\$ history

\$ clear

\$ locate

\$ open .

\$ exit

\$ kill

\$ head

\$ tail

\$ ps

# Initialise git

---

```
$ git init .
```

```
$ mkdir "hello git" && cd "$_"
```

```
$ git init .
```

```
$ ls -a
```

```
. . . .git
```

# Add file(s) to the staging area

```
$ git add <file>
```

```
$ git ls-files #like ls on stage
```

- ◆ <file> can be :
  - ▶ String : File name
  - ▶ • : all file within directory (Not recommended)
  - ▶ \* • [extension]

```
$ touch main.py
```

```
$ echo "print('hello world')\" > main.py
```

```
$ cat main.py
```

```
$ git add main.py
```

# Take snapshot (commit)

---

```
$ git commit -m "your commit message here"
```

```
$ git commit -m "Initial commit."
```

Change (last) commit message

```
$ git commit --amend -m "your new commit message"
```

You can commit and add on the same time (adds files already tracked to the staging)

```
$ git commit -am "your new commit message"
```

# Git log

Log history of commits

```
$ git log  
$ git log --oneline  
$ git log --oneline --reverse #start from the first commit
```

The screenshot shows a terminal window displaying the output of a `git log` command. The output lists two commits. The top commit is highlighted with a red box and has annotations pointing to its hash, message, and branch information. The bottom commit is partially visible.

```
[macbook-pro-de-amine:hello git amine]$ git log  
commit 61ad5ba39f8dccdae64b5b1a1825bbb2138cc0b6 (HEAD -> main)  
Author: AmineFrj <ferdjaouiamine@gmail.com>  
Date: Mon Nov 28 14:12:48 2022 +0100  
  
    Rename main.py  
  
commit fb90b43d2d78c2c709da49afe8e65e7d40e6d69f  
Author: AmineFrj <ferdjaouiamine@gmail.com>  
Date: Mon Nov 28 14:06:22 2022 +0100  
  
    add readme
```

Annotations:

- Commit hash**: Points to the commit ID `61ad5ba39f8dccdae64b5b1a1825bbb2138cc0b6`.
- Current branch**: Points to the branch name `(HEAD -> main)`.
- Commit message**: Points to the message `Rename main.py`.

# Git show & HEAD

---

View commit details

```
$ git show <hash>  
$ git show HEAD~N #The N-th commit before head  
$ git show HEAD~N:<filename>  
$ git ls-tree HEAD~N #like ls -a on certain commit
```

Head is a pointer to the a branch (branche's latest commit)\*

\*The HEAD could also be pointing to another commit and in that case it is called *Detached HEAD*

# Check changes

---

```
$ git status  
$ git status -s #cf. https://git-scm.com/docs/git-status
```

```
$ git status  
$ echo "print(\"Here is a new line\")" >> main.py  
$ git status  
$ git add main.py  
$ git status  
$ git commit -m "Add new print"
```

## Delete file(s)

---

Remove file(s) from working dir and in staging

```
$ git rm <file>
```

```
$ echo "test" >> README.txt  
$ git add README.txt  
$ git commit -m "Add a README."  
$ git rm README.txt  
$ git status  
$ git commit -m "Remove README"
```

# Delete file(s) from staging

---

Remove file(s) from staging

```
$ git rm --cached <file>  
$ git ls-files
```

```
$ touch main.py  
$ echo "print('hello world')\" > main.py  
$ cat main.py  
$ git add main.py
```

# Rename file(s)

---

```
$ git mv <old_name> <new_name>
```

```
$ git mv main.py model.py  
$ git status  
$ git commit -m "Rename main.py"  
$ git status
```

# Ignore files

---

Cf. <https://github.com/github/gitignore>

```
$ vi .gitignore
```

```
$ echo requirements.txt > .gitignore
```

Ps. Files created before .gitignore will not be ignored, you have to remove them from the staging

```
$ git rm --cached <file>
```

Then commit the changes

# View Staged differences

```
$ git diff #Difference between working dir and staging  
$ git diff --staged #Difference between staging and last commit
```

Last commit    Current staging

```
[macbook-pro-de-amine:hello git amine]$ git diff --staged  
diff --git a/main2.py b/main2.py  
index 7f58f33..411db58 100644  
--- a/main2.py  
+++ b/main2.py  
@@ -1,3 +1,4 @@  
print('hello world')  
print("Here is a new line \\!")  
print(Here is a new line)  
+#thisisacomment  
macbook-pro-de-amine:hello git amine$
```

Staging unique id ←

The 3 lines extracted from last commit ←

New line ←

This diagram shows the output of the command `git diff --staged`. It compares two versions of the file `main2.py`: the current staging area version (labeled "Current staging") and the version from the last commit (labeled "Last commit"). The output is a diff showing changes. Annotations with arrows point to specific parts of the output:

- An arrow points to the line `index 7f58f33..411db58 100644` with the label "Staging unique id".
- An arrow points to the first three lines of the diff output (`@@ -1,3 +1,4 @@` followed by the three printed lines) with the label "The 3 lines extracted from last commit".
- An arrow points to the line `+#thisisacomment` with the label "New line".