Version Control System

Git is a version control system (VSC). Version Control systems is used to track modifications, making a history of your code or files, collaborate with others and making sure that all of you on the same floor.

Git can be used locally and remotely (by GitHub).

Git workflow

Work Tree

The files you're working on and haven't been added to the staging area.

Staging Area

The files added from the working tree, ready to be committed to the git directory.

Git Directory

The directory of all your files including every commit with the status of your code in it.

Installation

One of the easiest to install on the three operating systems out there. Just go to the git site and choose the right configuration for your system and everything will be good.

Git Locally

Commands

Git init

First thing you want to do is to initialize a git work tree. It initializes empty repository.

Git status

Inform you of the status of the code (files changed and the current branch).

Git add <file_name> (may use ".")

Add certain or all files to staging area. And ready to be committed

Git commit

Commit the added files from staging area to git directory. Followed by -m to enclose a commit message.

Git reset HEAD <file_name> (may use ".")

Get back to undo the add command for a certain or all files.

Git checkout <file_name> (may use ".")

Get back to previous change before committing.

Git log
Show all your previous commits, with commits IDs to return to your wanted commit if needed.

Git diff
Show the changed to the working tree, of the files that hasn't been added yet.

Git rm <file_name>
Delete specific file from the git directory.

Git mv <old_file_name> <new_file_name>
Rename file names.

Git revert HEAD
Revert previous commit and undoing changes. Create a new commit with inverse changes.

Git revert <commit_ID>
Revert to specific commit and undoing changes.

Git branch
Show branches of repository you're working on right now.

Git branch <branch_name>
Create a new branch with that specific name. (Note: doesn't switch to it, just create)

Git switch <branch_name>
Switch to an existing branch.

Git checkout <branch_name>
Switch to an existing branch and get to the latest commit.

Git checkout -b <branch_name>
Create a new branch and switch to it. (Equivalent to git branch <branch_name> the git switch <branch_name>).

Git branch -d <branch_name>
Delete a branch locally. (If there are changes to the current branch that haven't been merged to the master branch, git will tell you, and if you're sure that you want to delete it, make it -D).

Git merge <branch_name>
Merge the specified branch to the current branch (Like you are on the master branch and want to merge another branch to it).

Git merge –abort
Cancels the merge and start over from the previous commit. (Useful in the merge conflict cases).

Git log --graph --online
Shows the history of commits in master and other branches.

Git Remotely

Commands

Git clone <repo_url>

Clone an existing repo to your computer. (Will ask for user and password if not provided already).

Git push

Pushes the files in the Git repo to the GitHub repo.

Git push -u <name> <branch_name>

Creating a remote branch (after creating it locally), then pushes to it.

Git pull

Pulls the files from remote branch and merges it with the local branch.

Git push –delete <name> <branch_name>

Delete branch remotely, the <name> here represents the name you used in pushing the very first time. (In most cases its name is "origin").

Best Practice

- *Avoid having big changes, try to make changes as small as possible.*
- *When working, it's best to have a separate branch. Then merge it to master.*
- *Regularly merge the changes made in the master branch to your branch.*
- *You can work on separate files from the start. Then merge everything.*
- *Make commit message meaningful.*
- *Everyone should agree on one way of writing code, like naming variables and functions.*
- *Any hardcoded values should be in a constant in the same file or separate file.*
- *Make a virtual environment to every project, since it doesn't conflict with existing installed packages.*

Notes

.gitignore

It's a file that contains specific files or specific extensions that you don't want to include in your git work flow.

requirements.txt

A file contains all needed packages in the project. Better with the exact version of every package to avoid package changed behavior.

Merge Conflict

Happens when there are changes to the same file in the same part. After git merge, git will tell you that there is a merge conflict. Requesting to fix the conflict first. Just open the file and choose what to do with, then add the file then commit.

Although it isn't always that easy, but it could help sometimes.