Git

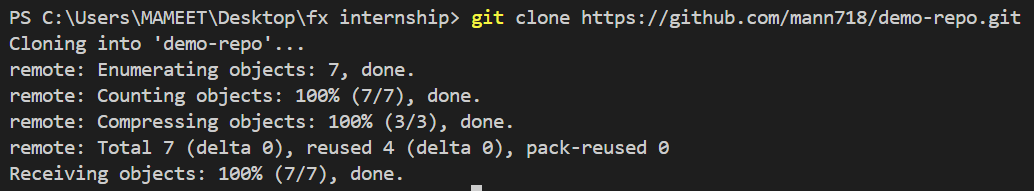
Why use Git?

Git is a version control tool. It helps software teams manage changes to source code over time. It is different from Github, Github is a website to host repositories online.

Git Commands

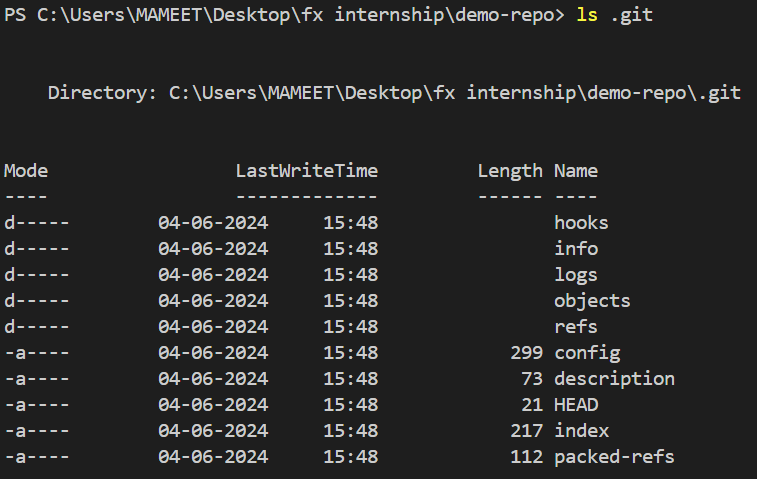
#### Git Clone

Clones existing git repository to local machine.



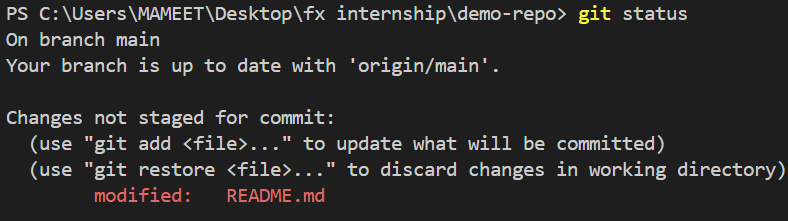
### .git Directory

.git directory makes folder structure a git repository. It introduces git functionalities to the project.



#### Git Status

The git status command displays the state of the working directory and the staging area. It lets you see which changes have been staged, which haven’t, and which files aren’t being tracked by Git.

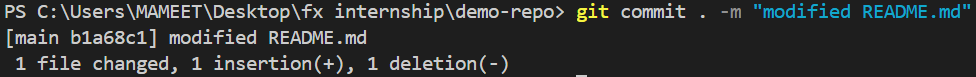


Git add

Moves changes from the working directory to the staging area. This gives you the opportunity to prepare a snapshot before committing it to the official history.

Git commit

Takes the staged snapshot and commits it to the project history. Combined with git add, this defines the basic workflow for all Git users.



Git commit -amend

Let's say you just committed and you made a mistake in your commit log message. Running this command when there is nothing staged lets you edit the previous commit’s message without altering its snapshot.

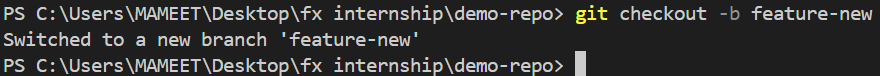
Git branch

This command is your general-purpose branch administration tool. It lets you create isolated development environments within a single repository.



Git Checkout

It is the means to navigate existing branches and even create new one.



### Git clean

Removes untracked files from the working directory. This is the logical counterpart to git reset, which (typically) only operates on tracked files.

Git fetch vs Git Pull

When downloading content from a remote repo, git pull and git fetch commands are available to accomplish the task. You can consider git fetch the 'safe' version of the two commands. It will download the remote content but not update your local repo's working state, leaving your current work intact. git pull is the more aggressive alternative; it will download the remote content for the active local branch and immediately execute git merge to create a merge commit for the new remote content. If you have pending changes in progress this will cause conflicts and kick-off the merge conflict resolution flow.

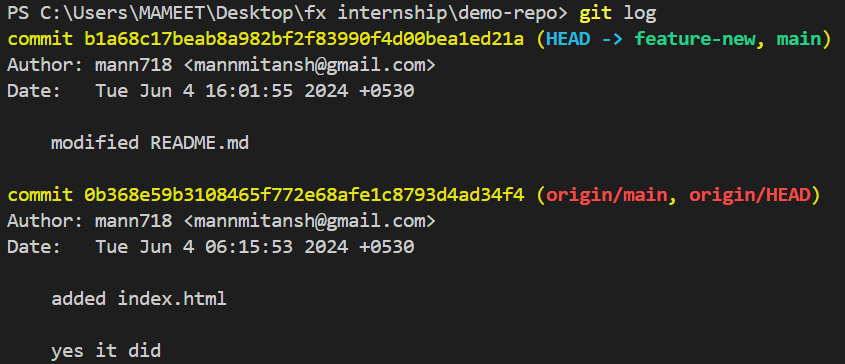
Git init

Initializes a new Git repository. Also can put previous project under revision control.

It creates .git directory.

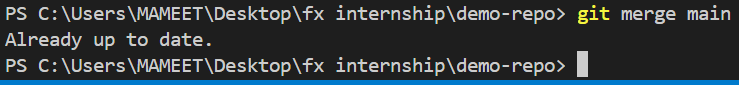
Git log

Lets you explore the previous revisions of a project. It provides several formatting options for displaying committed snapshots. It basically provides hash ids for different revision to revisit.



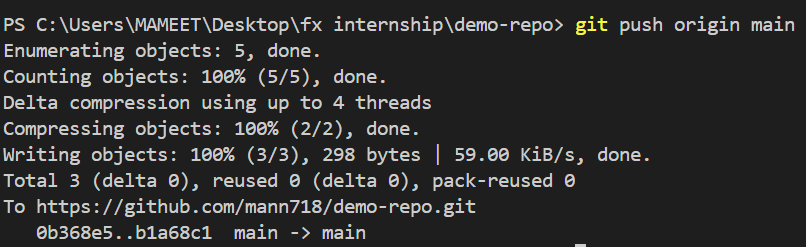
Git Merge

Invoking this command will merge the specified branch feature into the current branch

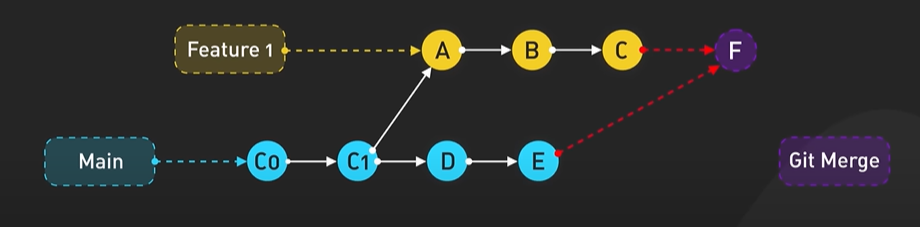


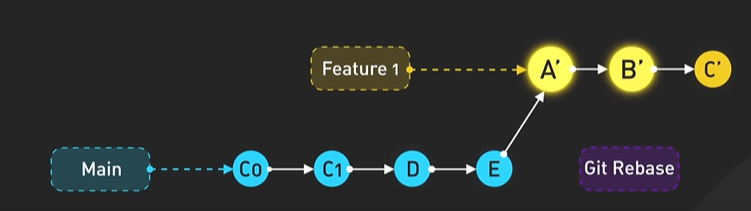
Git push

It lets you move a local branch to another repository, which serves as a convenient way to publish contributions.



Git rebase





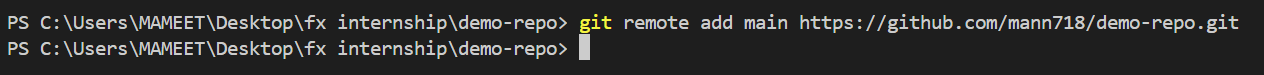
Rebasing is the process of moving or combining a sequence of commits to a new base commit. Rebasing is most useful and easily visualized in the context of a feature branching workflow.

### git rebase -i

The -i flag is used to begin an interactive rebasing session. This provides all the benefits of a normal rebase, but gives you the opportunity to add, edit, or delete commits along the way.

Git Remote

The git remote command lets you create, view, and delete connections to other repositories.



Git reset

Undoes changes to files in the working directory. Resetting lets you clean up or completely remove changes that have not been pushed to a public repository.

*PS C:\Users\MAMEET\Desktop\git\_clone\demo-repo> git status*

*On branch quick-branch*

*nothing to commit, working tree clean*

*PS C:\Users\MAMEET\Desktop\git\_clone\demo-repo> git reset*

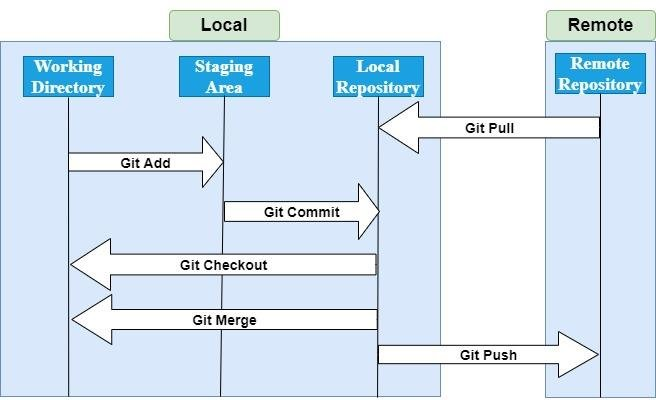
*PS C:\Users\MAMEET\Desktop\git\_clone\demo-repo> git reset README.md*

*PS C:\Users\MAMEET\Desktop\git\_clone\demo-repo> git reset HEAD~1*

*Unstaged changes after reset:*

*M README.md*

*General workflow with Git*



Generating and Setting up SSH keys

*MAMEET@LAPTOP-88LVITT6 MINGW64 ~*

*$ ssh-keygen -t rsa -b 4096 -C "mannmitansh@gmail.com"*

*Generating public/private rsa key pair.*

*Enter file in which to save the key (/c/Users/MAMEET/.ssh/id\_rsa): testkey*

*Enter passphrase (empty for no passphrase):*

*Enter same passphrase again:*

*Your identification has been saved in testkey*

*Your public key has been saved in testkey.pub*

*The key fingerprint is:*

*SHA256:uEwbN6N7Y8sDzmJ6m9Wjze7Ftcs8gS8k8VWV0R2uA0s mannmitansh@gmail.com*

*The key's randomart image is:*

*+---[RSA 4096]----+*

*| +B|*

*| o.o|*

*| E . . |*

*| .. . + . |*

*| + So +.o |*

*| o.B.++.... |*

*| o\*.ooo... |*

*| +oo\*=o.oo. |*

*| .+ooo=B+ .+. |*

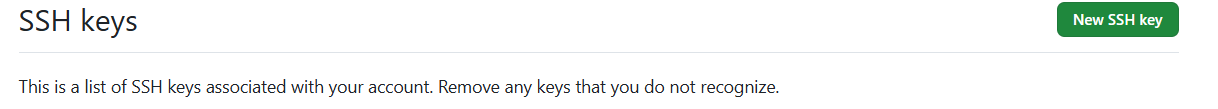
*+----[SHA256]-----+*

*MAMEET@LAPTOP-88LVITT6 MINGW64 ~*

*$ ls | grep testkey*

*testkey*

*testkey.pub*



Git Forking

A fork is a new repository that shares code and visibility settings with the original “upstream” repository.

For example, you can use forks to propose changes related to fixing a bug. Rather than logging an issue for a bug you have found, you can:

* Fork the repository.
* Make the fix.
* Submit a pull request to the project owner.

# **Python Virtual Environments**

Need of it.

The short answer is that Python isn’t great at dependency management. If you’re not specific, then pip will place all the external packages that you install in a folder called site-packages/ in your base Python installation.

You may need administrator privileges on a computer to install packages into the host Python’s site-packages directory. In a corporate work environment, you most likely won’t have that level of access to the machine that you’re working on.

If you use a separate virtual environment for each of your projects, then it’ll be more straightforward to read the project requirements from your pinned dependencies.

### Create It



Activate-Deactivate



## **What Is a Python Virtual Environment?**

It is a folder Structure. When you create a new virtual environment using the venv module, Python creates a self-contained folder structure and copies or [symlinks](https://en.wikipedia.org/wiki/Symbolic_link) the Python [executable files](https://en.wikipedia.org/wiki/Executable) into that folder structure.

venv\

│

├── Include\

│

├── Lib\

│ │

│ └── site-packages\

│ │

│ ├── \_distutils\_hack\

│ │

│ ├── pip\

│ │

│ ├── pip-22.0.4.dist-info\

│ │

│ ├── pkg\_resources\

│ │

│ ├── setuptools\

│ │

│ ├── setuptools-58.1.0.dist-info\

│ │

│ └── distutils-precedence.pth

│

│

├── Scripts\

│ ├── Activate.ps1

│ ├── activate

│ ├── activate.bat

│ ├── deactivate.bat

│ ├── pip.exe

│ ├── pip3.10.exe

│ ├── pip3.exe

│ ├── python.exe

│ └── pythonw.exe

│

└── pyvenv.cfg

## **What Other Popular Options Exist, Aside From venv?**

1. Virtualenv is a superset of venv and provides the basis for its implementation. It’s a powerful, extendable tool for creating isolated Python environments.
2. Conda offers package, dependency, and environment management for Python and other languages.