**Git and GitHub**

* Git is a version control system which can allow us to –
* Track the changes, who made the changes, and when
* Rollback/restore changes
* Help multiple developers to coordinate and work on the same files
* Maintain a copy of the files at the remote and local level

**How can Git be used in MLOps?**

* If I am required to track changes to my ML model, code, and configuration files, I can make use of Git as it allows me to track the code changes as well as restore to the code version or experiment version of my choice
* Git in conjunction with MLflow will be helpful to track and manage machine learning experiments. This includes recording parameters, metrics, and code versions, making it easier to reproduce and compare results
* Git’s branching feature allows for the creation of isolated branches to experiment with different model versions or parameter settings. This supports a systematic approach to testing and refining models without affecting the main codebase
* Idea is to utilize git’s capabilities in traceability and reproducibility of the ML code\

**What is a repository?**

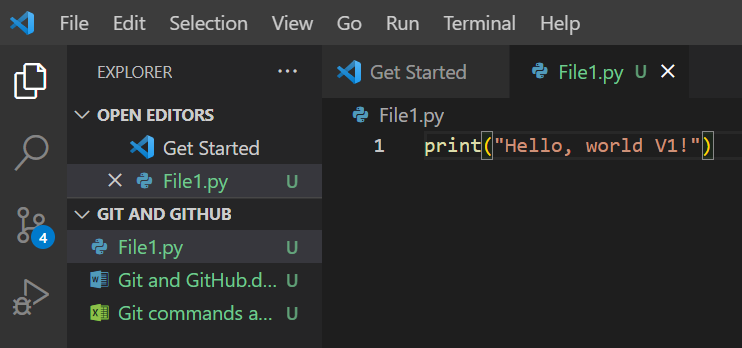
* A repository is set of project files which are version controlled by Git

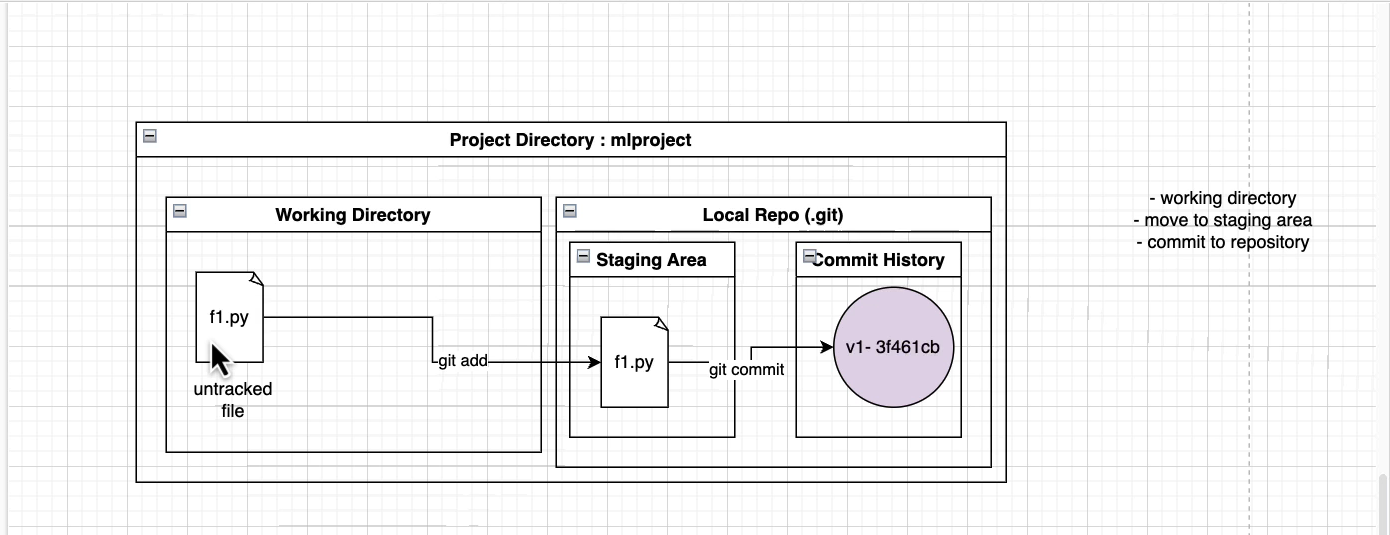
**What happens behind the scenes in git?**

I create a file. Initialize the git repository. Move the file to the staging area by running git add. Then run git commit when I run git commit the .git stores the current snapshot of the file and create a reference (HEAD) that points to the latest commit object. Next time if I make any changes to the contents of the file. Then I will have to again run git commit which will allow git to capture the screenshot of the file contents inside the .git file and the HEAD will now point to the latest commit. and if I want I can rollback to the very first version of the file because I have the snapshot saved in my .git file.

* **Configuration settings**
* Git configuration are settings that allow you to customize your global configuration file. They consist of variables and their values, and they are stored in a couple of different files. Basically, Imagine a global configuration file as your personal Git profile. It stores your preferences and settings that you want applied consistently across all your projects, just like a social media profile reflects your identity across different platforms. This centralized approach saves you time, improves efficiency, and allows for a personalized Git experience.
* After configuring my git profile, I will initialize git so that a .git file is created. This file is hidden which records the version of my project, kind of like capturing the various screenshots of my project. Please note *Git doesn’t literally take screenshots, it does create a versioned history of your project files, allowing you to track changes over time, roll back to previous versions, and collaborate with others*
* **Working directory**
* In Git, the **working directory** (or working tree) refers to the current state of the files in your project directory as seen by you, the user, on your local machine. It contains the actual files you are working on at any given time
* So, the working directory is essentially your active workspace where you modify and update files before using Git commands to track and manage those changes.
* **Staging area**
* The **staging area** is essentially a **draft space** that holds the changes you want to commit. When you modify files in the working directory, those changes don't automatically go into a commit. You have to add them to the staging area using git add. Once there, Git knows you are ready to commit these specific changes.
* **Commit**
* A **commit** in Git is like telling Git to **permanently record the changes** that you've staged
* **In summary:**
* **Staging area**: Prepares the changes (like a draft) that you want to commit.
* **Commit**: Finalizes and records those changes in Git’s history.

It’s like capturing a moment in time of your project so you can refer back to it later.

* **State of Files in the Working Directory:**
* **Unmodified**: Files that are the same as the last commit.
* **Modified**: Files that have been changed but not yet staged.
* **Staged**: Files that have been added to the staging area and are ready to be committed.
* **Git hands-on**
* 
* Notice in the image above, there is a “U” mentioned next to the name of the file in the working directory titled as “GIT AND GITHUB”.
* ***Interpretation of the above scenario***
* In Git, when you see a U next to a file in VS Code, it typically means that the file is untracked.
* **What "Untracked" Means:**
* Untracked files are files that exist in your working directory but are not yet being tracked by Git. In other words, Git is aware that the files exist, but it hasn't been instructed to keep track of their changes.
* This happens because you haven't staged these files yet using git add.
* **Using git status**:
* 
* As it can be seen from the output of the command above, 3 files are untracked
* **Using git add** – This will move the file from working directory to staging area



***Pictorial representation of what I have learnt so far***

* **Git branch**: