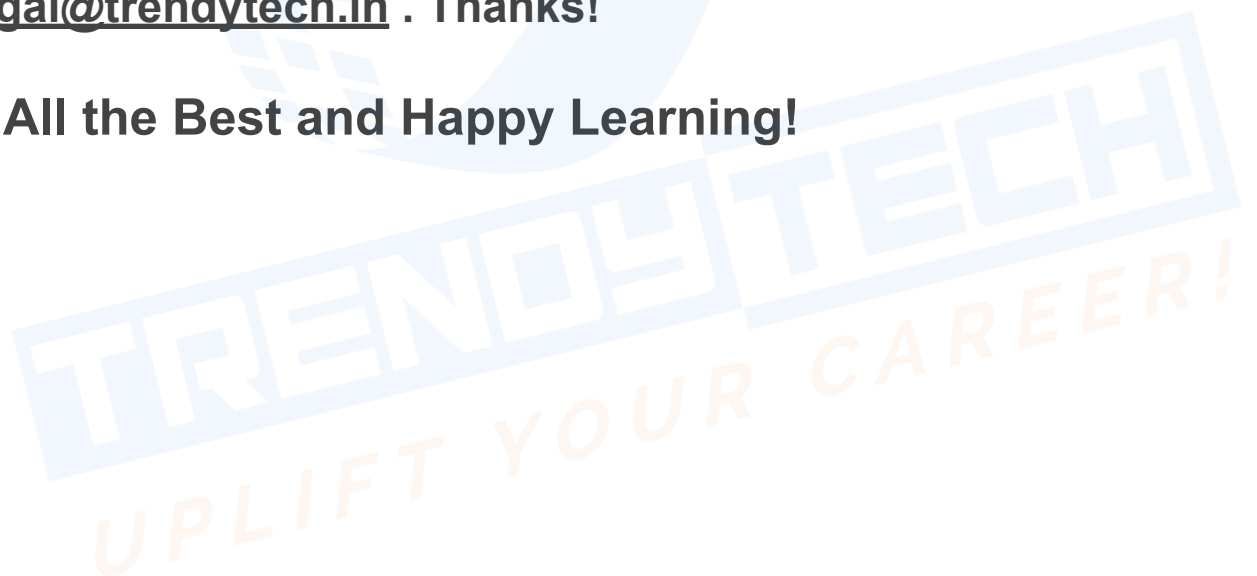


Disclaimer: These slides are copyrighted and strictly for personal use only

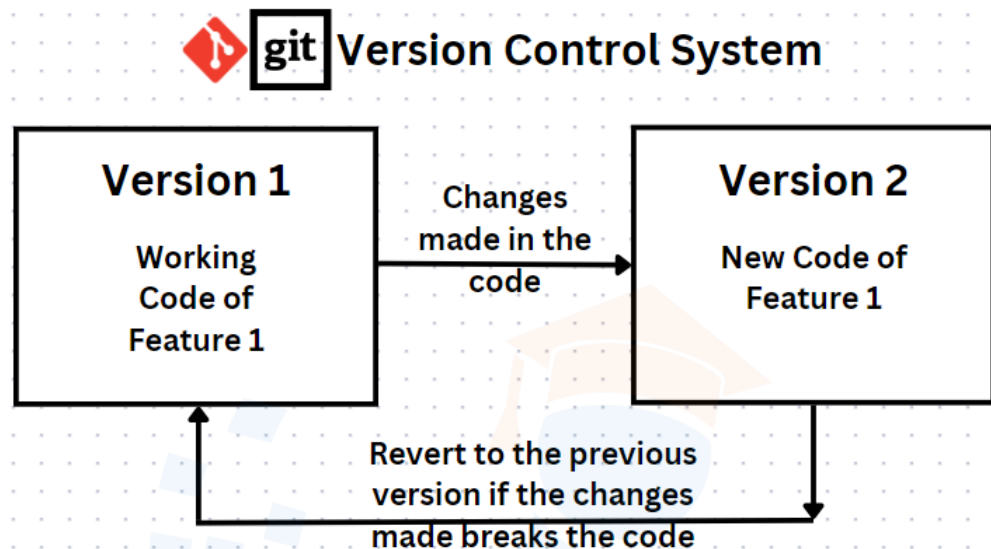
- This document is reserved for people enrolled into the [Ultimate Big Data Masters Program \(Cloud Focused\) by Sumit Sir](#)
- **Please do not share this document**, it is intended for personal use and exam preparation only, thank you.
- If you've obtained these slides for free on a website that is not the course's website, please reach out to legal@trendytech.in . Thanks!
- All the Best and Happy Learning!



Git & GitHub

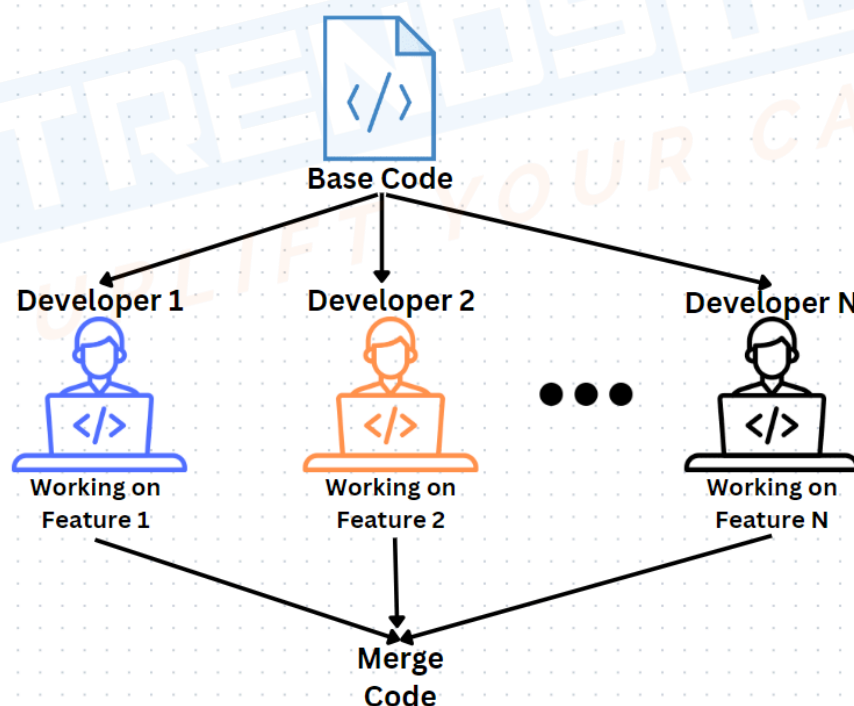
1. **Git (Local)** is a version control system for versioning the code

It is easier to revert back to the previous functioning version.



2. **Git is used for Collaboration**

Multiple developers work on the base code to develop different features and all of the code is merged to get the final code with the required features.

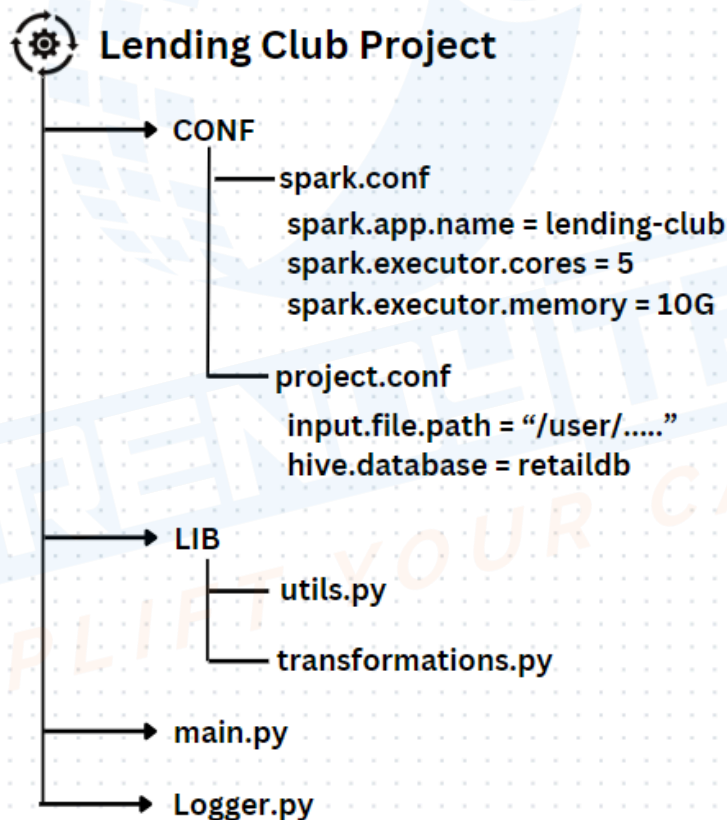


GitHub (Remote)

It is an online platform for storing, tracking and collaborating on projects. It is like a website where you host the projects.

Basic Project Structure in Pyspark

- All the configuration information related to different environments like Development, Production, Testing will be present under the **Conf** folder.
- Any reusable code is stored under the **Lib** (Library) folder to make the code modular and reusable.
- All the login related details are stored in **Logger.py** file
- **main.py** (wrapper file) is an entry point to the application, this file consists of the execution sequence. Various functions (transformations & Actions) are called from the main file.



Creating a Project in your Local System (Local Repository)

You can use a command line editor like vi / vim to create the project structure as shown in the above diagram.

Note:

- MAC users can use the terminal and vi editor to create the project structure.
- Windows users can use the command prompt or power shell and vim editor to create the project structure in the local system

Steps for creating a project structure :

1. Create a project folder on your Desktop

```
cd desktop
```

```
mkdir lending-club-project
```

2. Create a main.py file within the project folder that acts as an entry point

```
cd lending-club-project
```

```
vi main.py
```

OR

```
vim main.py
```

(The code to call various functions required to achieve the desired logic will be present in main.py)

3. Create logger.py

```
vi logger.py
```

(This file will have all the login level details)

4. Create a conf folder

```
mkdir conf
```

```
cd conf
```

4a. Create a spark configuration file that will contain all the spark configurations

`vi spark.conf`

OR

`vim spark.conf`

4b. Create a Project configuration file that will contain all the project configurations

`vi project.conf`

OR

`vim project.conf`

Note :

- Ideally, to create a project structure as explained above, IDEs like Visual Studio / Pycharm / Eclipse will be used in real-time projects.
- lending-club-project folder created in the above steps is a **local repository** that is visible and accessible only on your local system. **(LOCAL - Git)**
- In order to make it visible and accessible to other project members and to be able to collaboratively work on the project, it has to be hosted online on a website and this can be achieved through **GitHub. (REMOTE - GitHub)**
- Apart from GitHub, there are other alternative online platforms like **BITBUCKET / GitLab**

Installing Git (Local Repository)

On Mac

`brew install git`

(This command would require Homebrew already installed on your system. If not installed, install Homebrew and then execute the above command to install git)

On Windows

Navigate to <https://git-scm.com/> and download the git installer executable file. Follow the installation wizard with the default settings and git will be installed.

To check for successful installation of Git, execute the following command

git --version

Note for Windows Users :

- git bash (linux terminal for windows) can be used in place of a terminal to execute the git commands.

Creating GitHub Account (Remote Repository)

Navigate to github.com and sign-up for account creation.

Scenario 1 : Creating a project on GitHub and cloning it to local


Step 1 : Start by creating a remote repository in GitHub

GitHub profile -> Repositories -> New

Create a new repository



A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk (*).

Owner *  manasa45 /

Great repository names are short and memorable. Need inspiration? How about [silver-pancake](#) ?

Description (optional)

- ☒  **Public**
Anyone on the internet can see this repository. You choose who can commit.
- ☐  **Private**
You choose who can see and commit to this repository.

Initialize this repository with:

- ☐ **Add a README file**
This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore

.gitignore template: **None**

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license

License: **None**

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

Step 2 : Get the remote repository content to your local system using the following command

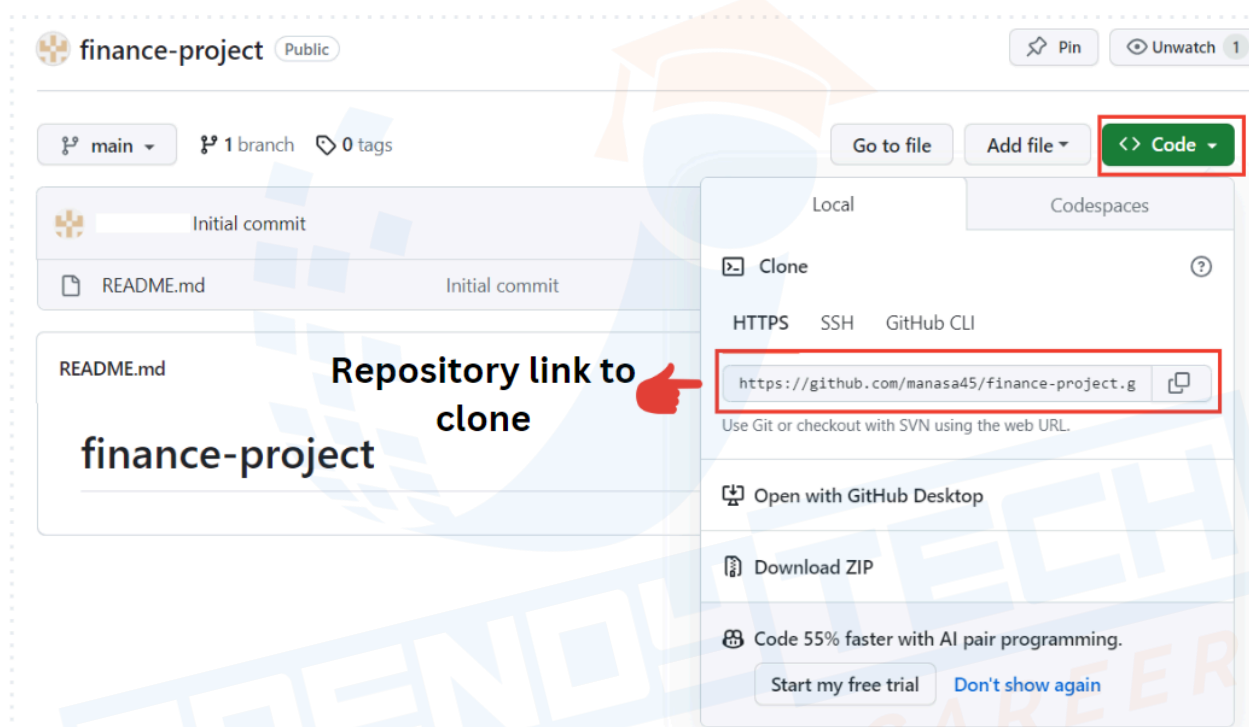
“Navigate to your Desktop in visual studio code terminal window and execute the following”

```
mkdir finance-project
```

```
cd finance-project
```

```
git clone <repository-link>
```

Example : `git clone https://github.com/manasa45/finance-project.git`



Important Git Commands

`git` (Gives a list of git commands)

`git --version` (displays the git version installed successfully)

`git init` (Initializing the empty repository)

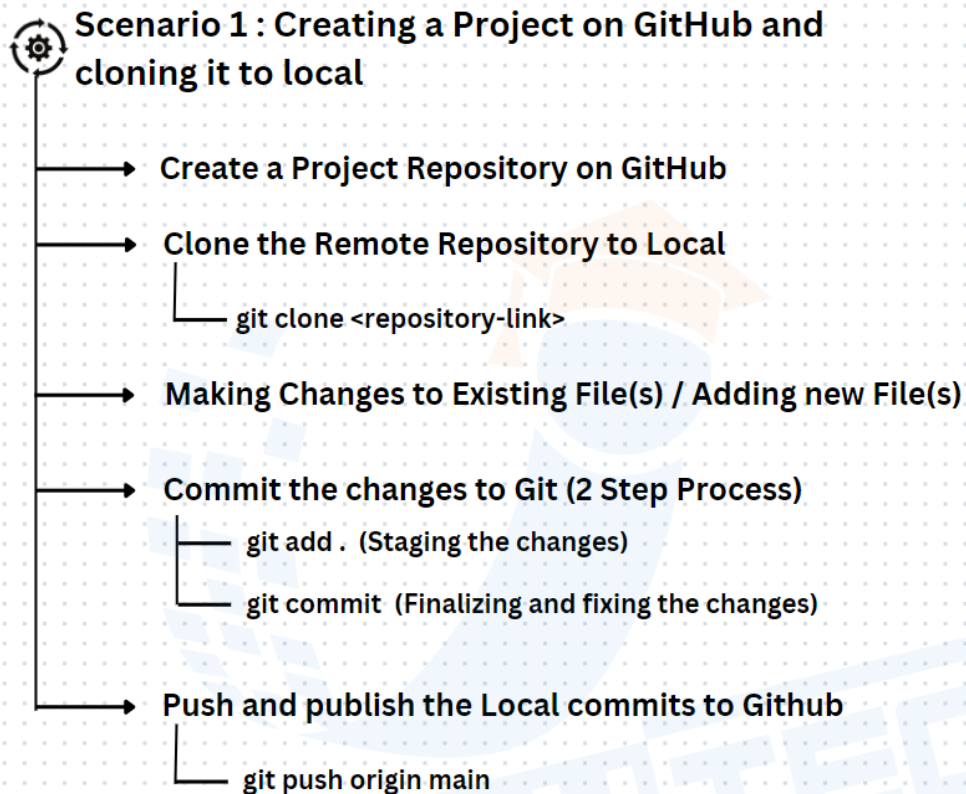
`git status` (Unstaged, Stage, Commit)

`git add <filename>` (Staging specific files)

git add . (Staging all the files that are changed)

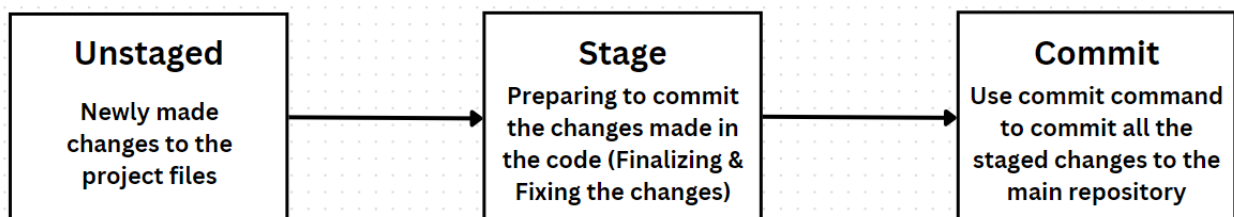
git commit -m "<message>" (Commit changes with appropriate message indicating the changes made)

git push origin main (pushing the local commits to the remote central repository)



Note:

- If new files are added to the project folder, it would reflect as untracked / Unstaged.
- If an existing file is changed, it would reflect as modified
- Unstaged changes need to be staged to commit the code changes to the central repository.



Scenario 2 : Creating a project locally on Git and adding it to the remote GitHub Repository

Step 1 : Create a Project Structure locally on your IDE

Step 2 : Since the local project created is not managed by Git, it is required to initialize git in the project folder using the following command

`git init`

Step 3 : Make the necessary changes and stage the changes to commit.

Step 4 : Commit the staged changes

`git add .`

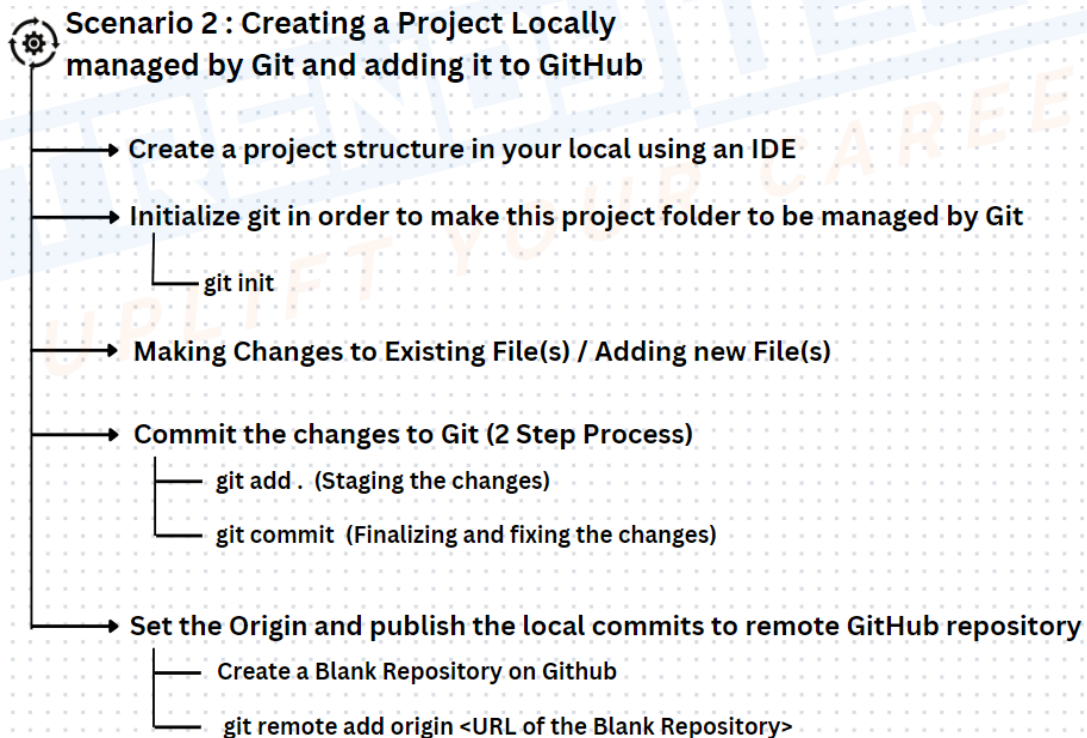
`git commit -m "<message>"`

Step 5 : Since this project is developed locally and not cloned from a remote repository, it is necessary to set the Origin before pushing the project to the remote gitHub repository.

To set the Origin

- Create a blank repository on GitHub
- And execute the following command with the URL of the remote repository created in the previous step. (**Rename the master branch to main**)

`git remote add origin <URL of the Blank Repository>`



Important Git Commands

git branch (displays all the existing branches)

git branch -M <new-name> (Renaming the branch)

git branch <branch-name> (Creates a new branch)

git checkout <branch-name> (Switching branches.)

git checkout -b <branch-name> (If the branch doesn't exist, it will create the branch and switch to the newly created branch)

git checkout -b <new-branch-name> <existing-branch-name> (Creates a new branch and takes the base code of the existing branch specified in the second parameter)

git branch -d <branch-name> (Delete branch is possible only if the current branch is not the same branch to be deleted. You have to checkout to a different branch before deleting the current branch you are in)

Scenario 2 : Making Changes to an existing Project

Step 1 : Create a Branch with the Base Code of any existing branch.

Each developer will create their own branch with the base code of a suitable existing branch to start with. Then build on this base code to develop the new feature or make changes to existing feature.

Step 2 : Push the newly created branch code to GitHub

Step 3 : Create a Pull Request to check the differences between the main branch and your newly created branch.

Step 4 : Review and Approval process to merge the changes to the main branch.

Step 5 : After the changes are merged to the main branch in GitHub, the changes don't reflect in your local git. In order to get your local in sync with the remote Github repository, execute the following

git pull origin main

Note :

Learn and explore more on branching - <https://learngitbranching.js.org/>

Reverting Back to the Previous Code Base

Say you have made changes to the code base and these changes broke the code with some errors. You would like to revert back to the previous functional code, how can this be achieved?

There could be 3 possibilities

Scenario 1 : Unstaged Changes

In this case since the changes are still not stages, you can use the following git command to revert back the changes.

```
git restore <file-name>
```

Scenario 2 : Staged Changes

In case you would like to revert back the staged changes, you need to follow 2 steps -

1. Unstage using the following command

```
git restore --staged <file-name>
```

2. Restore the unstaged changes in the previous step using the following command

```
git restore <file-name>
```

Scenario 3 : Committed Changes

In case you would like to revert back the committed changes, use the following commands

git log (Displays the commit hash value. Use this hash value in the reset command)

```
git reset <commit-hash-value>
```

(Each commit is associated with a hash-value that uniquely identifies the commit)

```
commit 20839e8426b4a85947cc04e7bea91650a875cf37 (origin/main, main, feature1)
```

git reset head~1 (To revert back the latest commit - One commit back)

git reset --hard <commit-hash-value> (reverts the changes completely without going through the unstaged step)

Scenario 3 : Contributing to an existing project (within an organisation / a open source project)

FORK command is used to get the copy of the code of an external project present in another GitHub account to your GitHub account, so you can work and contribute to the project.

- Once the project is added in your GitHub Repository, you will need to bring it to your local system to start working on the project.

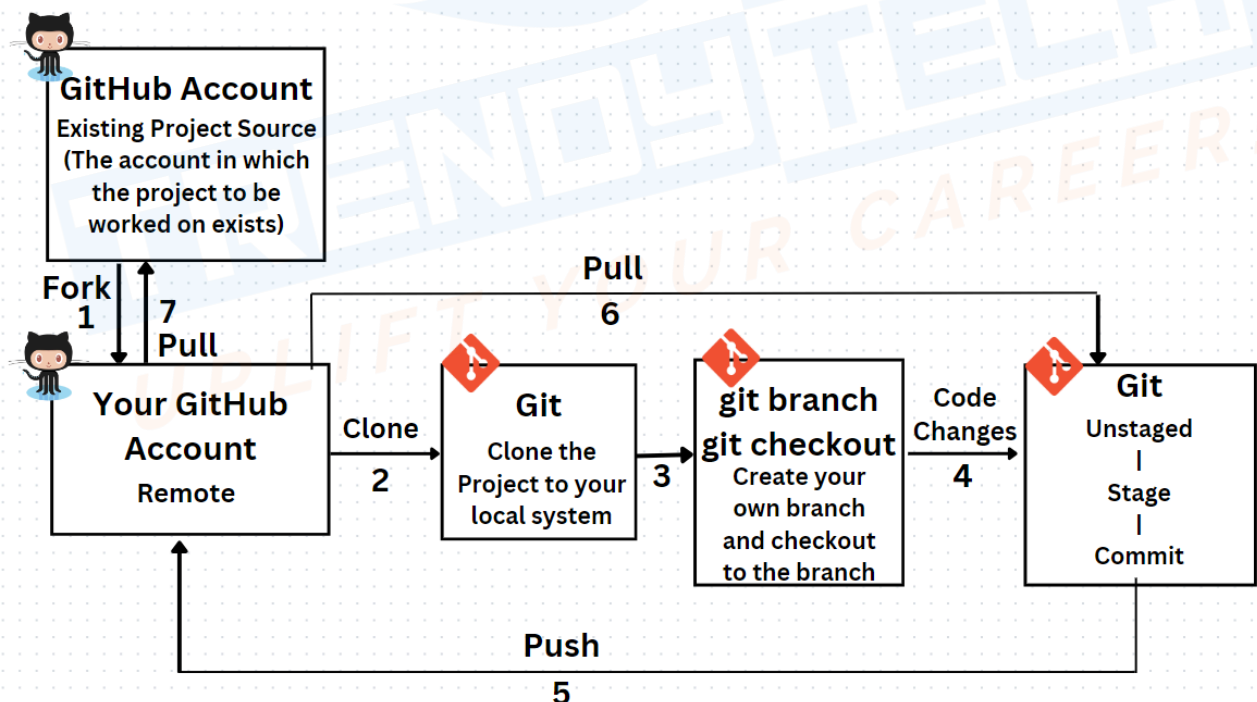
Firstly, you will have to **create a blank project** on your local system.

Secondly, use the **git clone** command to bring the project from the remote repository to local.

Thirdly, create a branch and checkout to your branch to make the changes. (Changes should not be made directly to the main branch)

Fourthly, make the necessary changes or additions. Commit the changes made and push the changes made to the Remote repository for approval process.

Fifthly, once changes are reviewed and approved, sync the local with the main branch with the pull command to keep the local up-to-date with the main branch



Important Git Commands

git remote add upstream <upstream-URL> (Setting the upstream URL)

git remote -v (Displays the information of where the origin and upstream is pointing)

git pull upstream main (syncing the code with the upstream main branch)

git stash command used to store/park the changes made in the code for later retrieval as you don't want these changes to be effective at the moment.

git stash clear used to clear the stash which permanently deletes the changes that are parked for later retrieval.

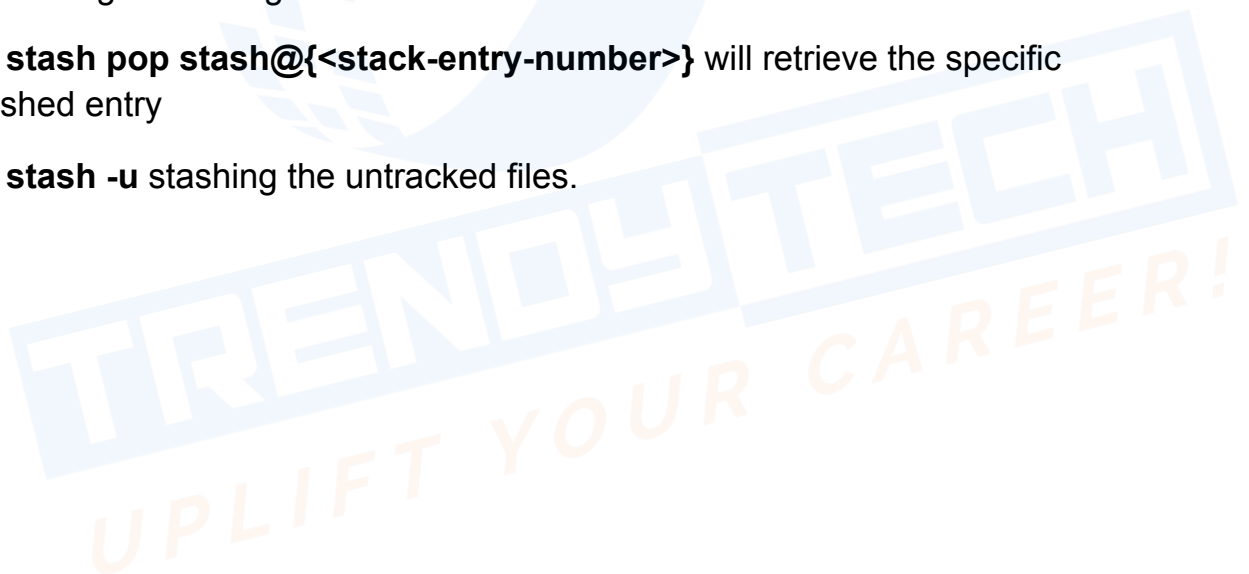
git stash list displays the list of changes made and stashed one after the other.

git stash pop will retrieve only the latest made change

git stash save "<message>" will store the changes made by associating with a meaningful message for easier retrieval.

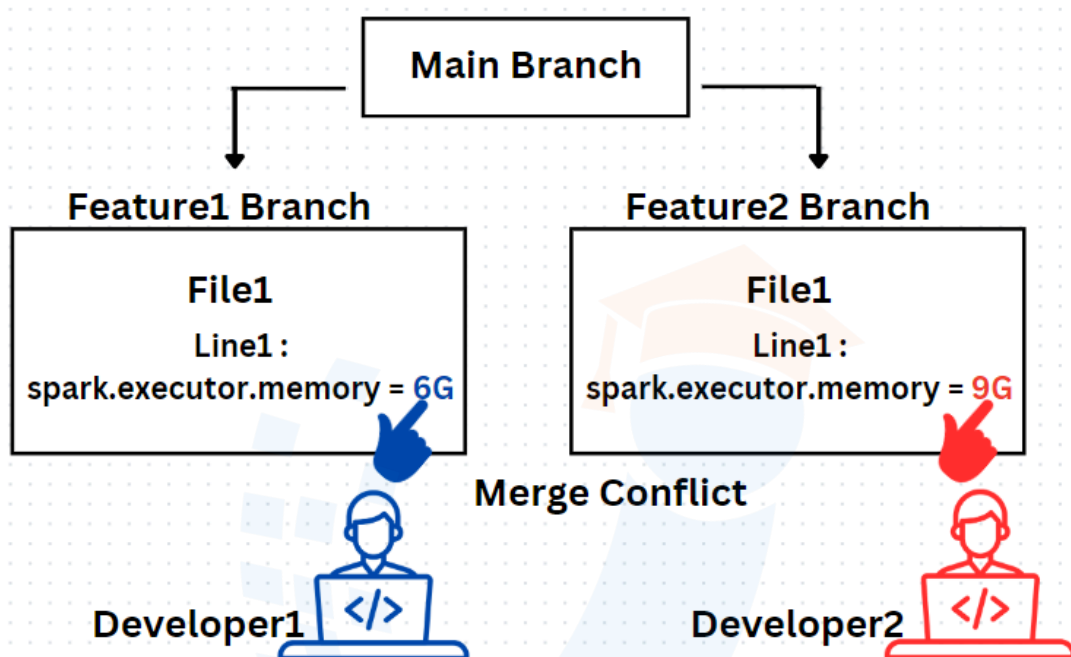
git stash pop stash@{<stack-entry-number>} will retrieve the specific stashed entry

git stash -u stashing the untracked files.



Handling Merge Conflicts

Conflicts occur in merging when git is not able to identify the changes made.



When pull request is initiated in GitHub, the approval team will check for the merge conflicts and resolve the conflicts by retaining the required code changes and eliminating the not required code lines that are causing the conflict.

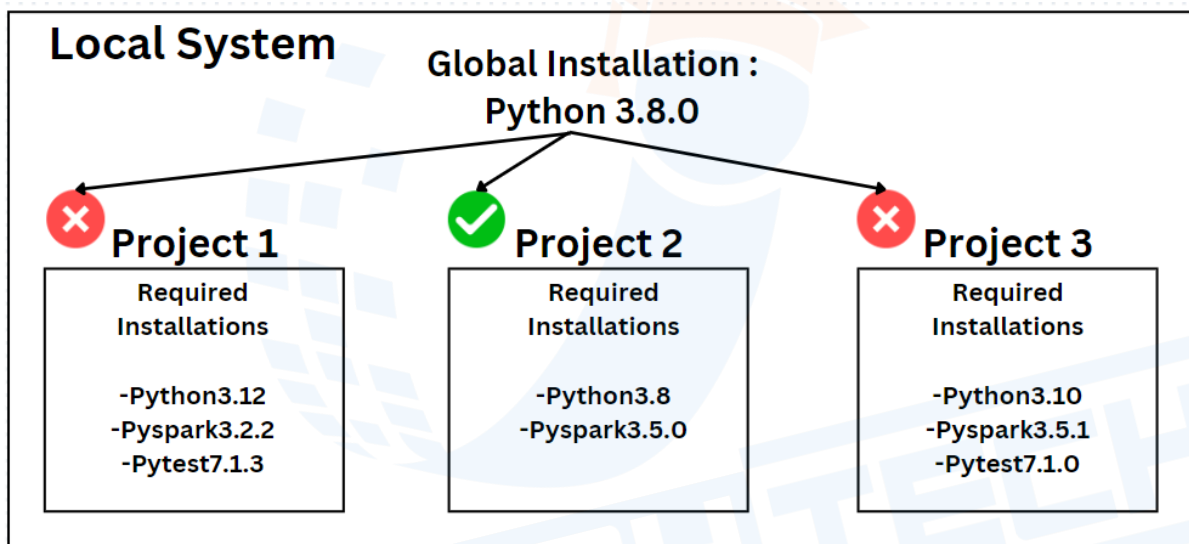
Project Structuring

Notebooks are for exploration and developing logic.

In the industry, developers use IDEs - an integrated environment for developing code that has several features supporting and easing code development. Ex - Visual Studio Code, Pycharm

How to Setup the project on Local System

Say you are working on Multiple Projects and each project requires different versions of softwares installed. Then, having single global versions of softwares would not suffice the requirement of working on multiple projects.



Continuous Integration & Continuous Deployment - CICD

Source Control -

Every project will have a Central repository to store the source code. Ex - Github, Gitlab, Bitbucket

Consider a Project named - **Retail Analysis** and multiple developers are working on a project.

One of the Developers gets assigned a ticket in a ticketing tool like **JIRA** to develop code for a new feature that needs to be incorporated to the main project.

Ex - Jira Ticket : RA-17843

In order to work on the assigned task/ticket, the developer will create his own branch using the source control system and start developing the feature code locally. Once the code is developed, it will be pushed to the central repository for review.

Ex - Github : feature-RA-17843

(Feature branch is a short-lived branch. Once the code is reviewed and merged to the code base, the feature branch can be deleted, if not required.)

Branching Strategy

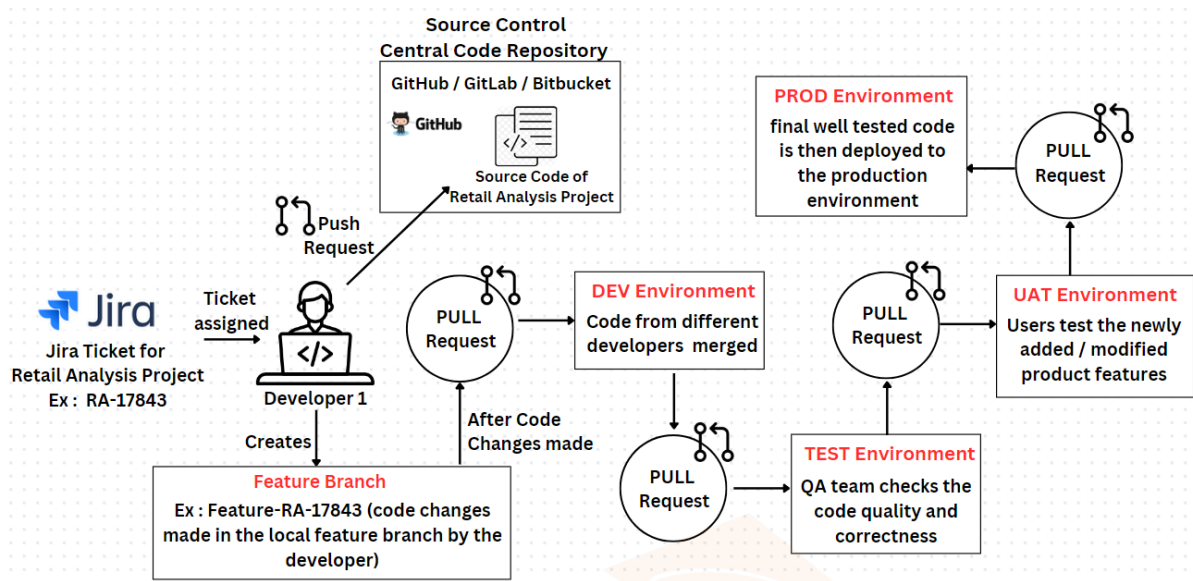
There are different environments in which various teams of a project work in. There could be a combination of the following environments depending on the size of the company.

DEV - Code from different developers will be merged and some basic code compatibility tests will be run to check if everything is working seamlessly.

TEST - Quality Assurance team will perform necessary tests to check for functional correctness.

UAT - Users perform tests after the changes are made to the product to analyze if the requirement is met.

PROD - final well tested code is then deployed to the production environment.



Java

Compile and Build -> Unit Tests -> JAR File created (known as Artefacts) -> Deployed (Placed in the server where the code can be executed using a secure copy command-scp.)

Python

1. **Build phase** - Create a Virtual Environment with all the dependencies installed
2. **Test phase** - DevOps team will perform **Unit Testing** and also **Check the Code Quality** based on the rules set in the pipeline.
3. **Package** - Zip File is created
4. **Deploy** - Place the code in the server using **SCP** command for execution

Note : This process of making the code changes, building, testing, packaging and deploying can lead to errors when there are multiple developers working on the project.

Therefore, it will be efficient to automate this process, wherein any changes done to the code will automatically go through all the above mentioned phases without any manual intervention.

This pipeline of building, testing, packaging and deploying is termed as CICD and can be automated.

CI - Continuous Integration - Build & Test

CD - Continuous Deployment - Package & Deploy

Automatically the complete CICD Pipeline should run in the following scenarios :

1. When a developer performs a GIT PUSH to a feature branch.
2. When a Pull Request is made from Feature -> DEV
3. When a Pull Request is made from DEV -> TEST
4. When a Pull Request is made from TEST -> UAT
5. When a Pull Request is made from UAT -> Main / PROD

The automation of the CICD pipeline requires an Automation Server

(**Jenkins** is one of the widely used automation servers)

Note : Pipeline is a series of steps that has to be performed one after another as per the requirement.

Deploying and Configuring Jenkins Server

(Usually these activities are performed by DevOps Engineers)

1. Go to cloud.google.com
2. Create a GCP account (provides 300\$ free credits that would be valid for 3 months)
3. Once you create an account -> Go to Console =>
<https://console.cloud.google.com>
4. Create a new Project and Deploy the required Resources (Like - Jenkins) and install the required packages.
5. Install the following plugins in the Jenkins UI account

- Dashboard View (Provides different views of the job - like grids, pie charts and so on)
- Github Branch Source
- Pipeline Declarative
- Pipeline Stage View

