# VCS (Version Control System)

## What is VCS?  Why we need VCS?

VCS is a software that is used to manage and control the versions of the source code.

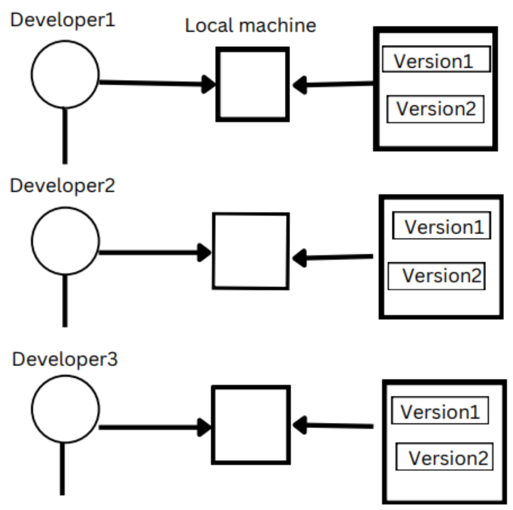
It will track (who? where? when?) all the changes made to the file.

## Types of VCS

1. **Localized VCS**
2. **Centralized VCS**
3. **Distributed VCS**

## Localized VCS

Using the local version control system developers will save the code/ files to the local machine/local repo. Best fit for individual users.



## Advantages of Localized VCS

1. Code will be saved to local machine so that we can refer the code. Developers can modify the code.
2. Developers will be able to switch back to the versions of file/code whenever they need.

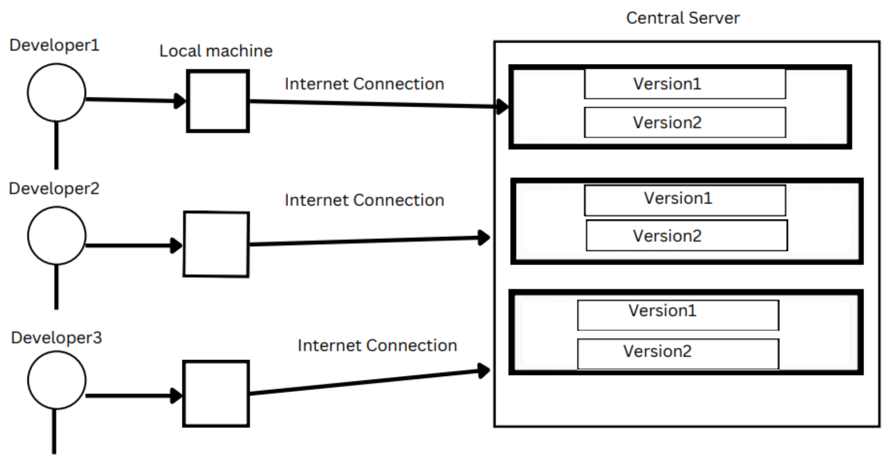
## Disadvantages of Localized VCS

1. If local machine fails then there will be loss of code.
2. Code integration is not possible.
3. No collaboration between developers.

## Centralized VCS

Centralized VCS allows developers to save the code in a central repository where all the developers can access the code with the help of the internet.

Example: Subversion, Perforce etc.,



## Advantages

1. Developers can access the code and they can do modifications to the code.
2. Code will be integrated.
3. Developers will collaborate with each other.

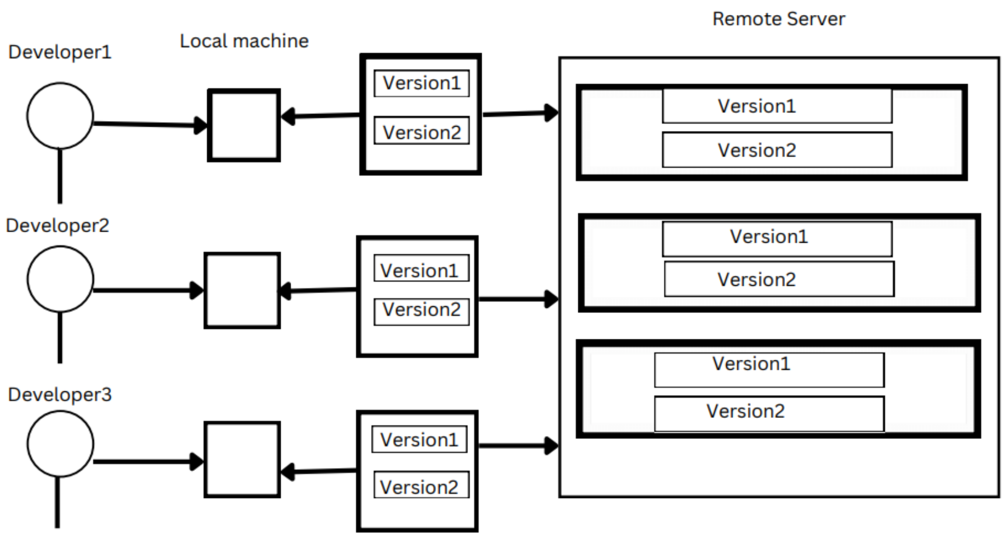
## Disadvantages

1. Internet is mandatory.
2. If central repo fails, then code is lost.

## Distributed Version Control System

This helps developers to save code in local machine and in central repository (remote server).

Ex: GIT, Mercurial etc.,



## Advantages

1. Code integration is possible.
2. Collaboration among developers is possible.
3. Internet is not mandatory.
4. If the local repo or central repo fails, then we can get code from any of the other developer’s local machine.

# GIT

## What is GIT?

GIT is a version control tool which follows Distributed Version Control System which is used to control and manage the source code.

## Features of GIT:

1. **Tracks history:** Git records every change made to files, allowing users to view and revert to previous versions as needed.
2. **Free and Open source:** Git is freely available for anyone to use, and its source code is open for inspection and modification.
3. **Supports non-linear development:** Git enables branching and merging, allowing developers to work on multiple features or fixes simultaneously without disrupting each other's progress.
4. **Creates backups:** Git repositories serve as backups of project code, ensuring that changes are not lost and can be restored if necessary.

5. **Supports collaboration**: Git facilitates collaboration among developers by providing tools for sharing and merging changes, as well as managing access control.
6. **Branching is easier:** Git branching model is lightweight and efficient, making it easy for developers to create, manage, and merge branches.
7. **Distributed Development:** Git allows developers to work independently on their own copies of a repository, then synchronize their changes with others, enabling decentralized and distributed development workflows.

## History Of GIT

- Introduced in the year 2005.
- Introduced by Linus Torvalds.

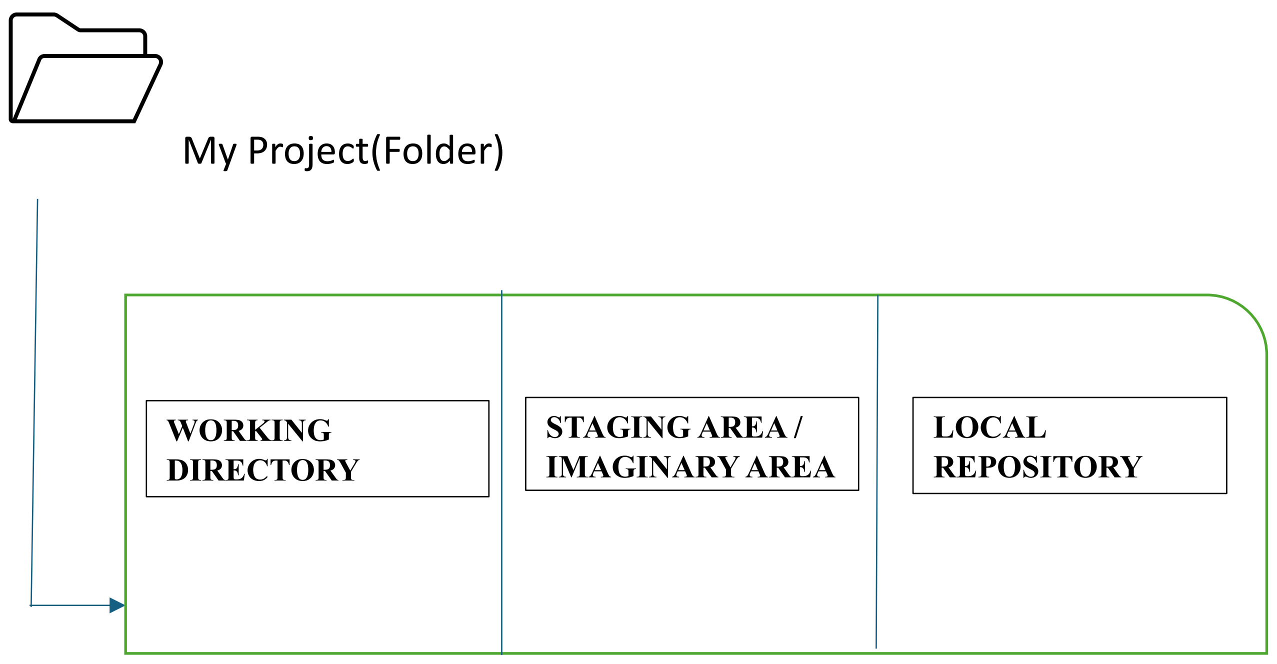
## Working Stages of Git:

There are 3 Working Stages in GIT.

1. Working directory.

2. Staging area.

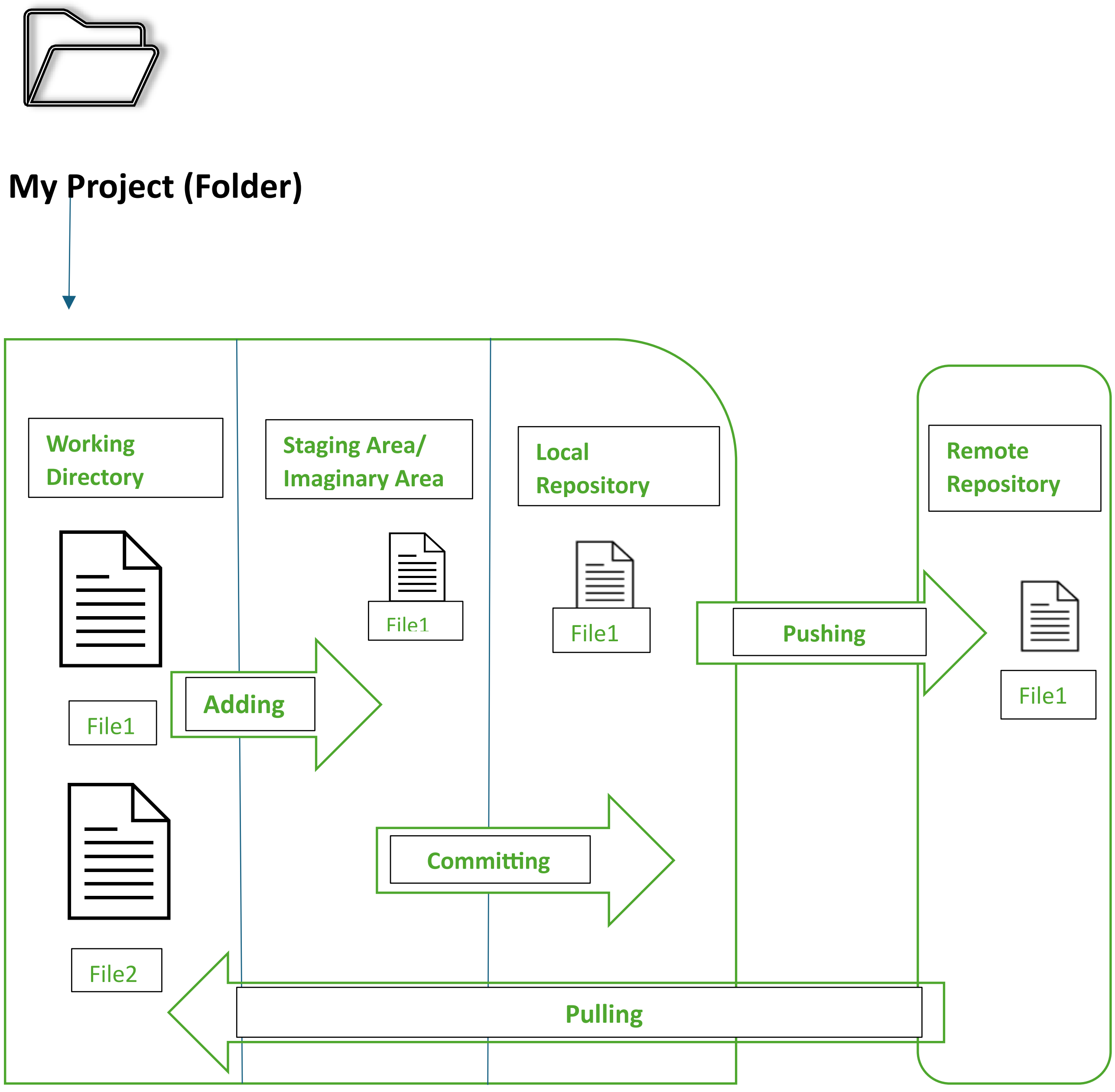
3. Local Repository.



**My Project (Folder)**

- Create a folder in local device (My Project).
- Do initialization to initialize GIT into local folder.
- After initialization **.git** folder is created inside a folder (My project) which is hidden inside the folder.

## Workflow of GIT:



## Workflow of GIT:

There are 7 stages in workflow:

1. Initialization
2. Configuration
3. Adding
4. Committing
5. Connection
6. Pushing
7. Pulling

1. **Initialization:** To initialize the GIT into local folder.
2. **Configuration:** We need to configure the user name and email address to identify who is modifying the source code and to contact the user if any changes should be made.
3. **Adding:** Moving files from working directory to staging area.
4. **Committing:** Moving files from staging area to local repository by committing.
5. **Connection:** Establishing connection between local repository and remote repository.
6. **Pushing:** Copying files from local repository to remote repository.
7. **Pulling:** Copying files from remote repository to working directory.

## Basic Commands to perform above operations.

1. To initialize the local repository.
   - **git init**

2. To configure the user information.
   - **git config user.name “ABC”**

- **git config user.email “abc@gmail.com”**

To setup username and email globally

- **git config - -global user.name “abc”**
- **git config - -global user.email “abc@gmail.com”**

3. Adding files from working directory to staging area.
   - **git add filename** → to add single file.
   - **git add .** → to add all the files.

4. Committing files from staging area to local repository.
   - **git commit -m “any message”**

5. To establish connection between local and remote repository.
   - **git remote add [alias name**→origin**] URL** (remote URL)

6. To push files from local to remote repository.
   - **git push origin master** (master is default user branch)

7. To pull the files from remote repository to local repository.
   (pull is the combination of fetch and merge)
   - **git pull origin master**

## Fetch commands:

- **git fetch origin** -> To fetch the files from remote repository to the local repository.

## Extra commands:

- To check files are added into staging area.
  - **git status**

- To check commit history.
    - **git log**

- To check git version.
    - **git - -version**

- To check connection is established between remote and local server.
    - **git remote -v**

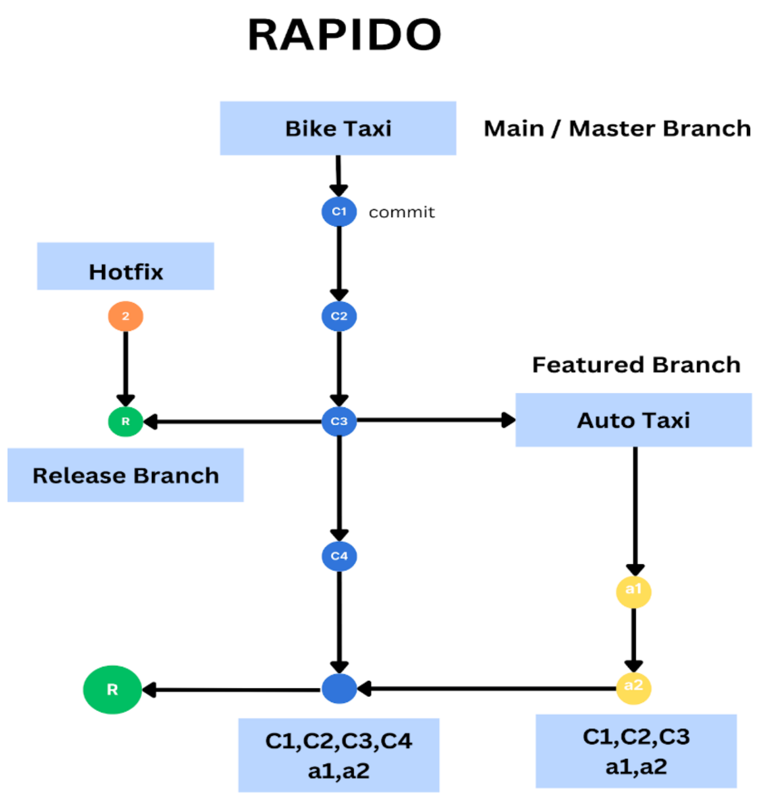
- To edit the commit message.
    - **git - -amend**

- To remove file from local repository.
    - **rm filename**

- To remove file from remote repository.
    - **git rm - -cached filename**

- To edit email and username.
    - **git config - -edit.**

## Git Branches



- **Master Branch:** Often called master or main, this is the primary branch of your project. It typically represents the stable version of your code.

- **Featured Branch:** These branches are created to implement new features or functionalities. They are usually created off the main branch and merged back into it once the feature is complete.

- **Release Branch:** Created when preparing a new release version of the software. It's used for finalizing the release, making last-minute changes, and ensuring stability before merging into the main branch and tagging for release.

- **Hotfix Branch:** Sometimes, issues or bugs arise in the main branch that need to be fixed urgently. Hotfix branches are created from the main branch to address these problems quickly. Once fixed, they are merged back into both the main branch and any active development branches.

## Branching commands

- **git branch branchname** -> To create branch

- **git switch branchname** -> To switch branch

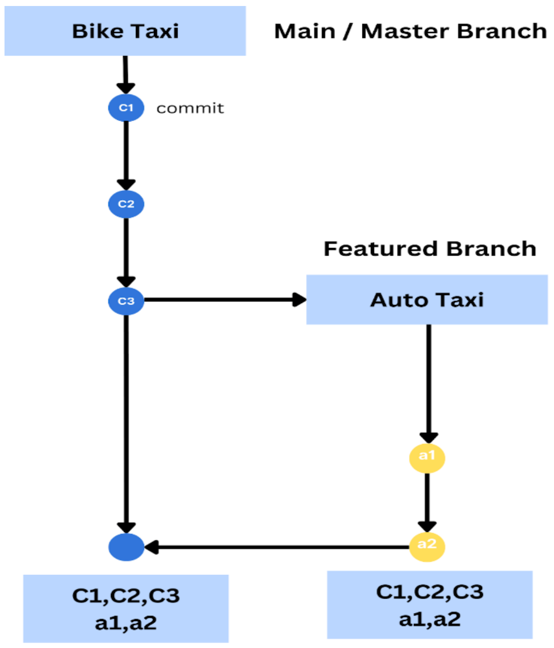
- **git checkout branchname** -> To switch branch

- **git checkout -b branchname** -> To create and switch to that branch.

- **git branch -d branchname** -> To delete the branch locally.

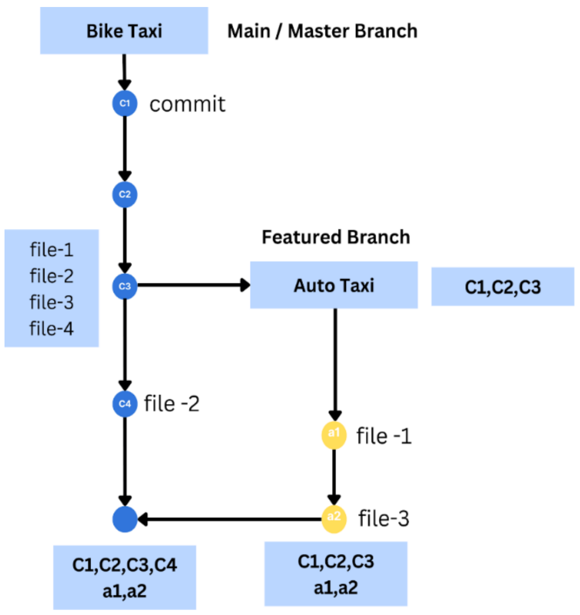
# Merging Branches

## 1. Two-way merging



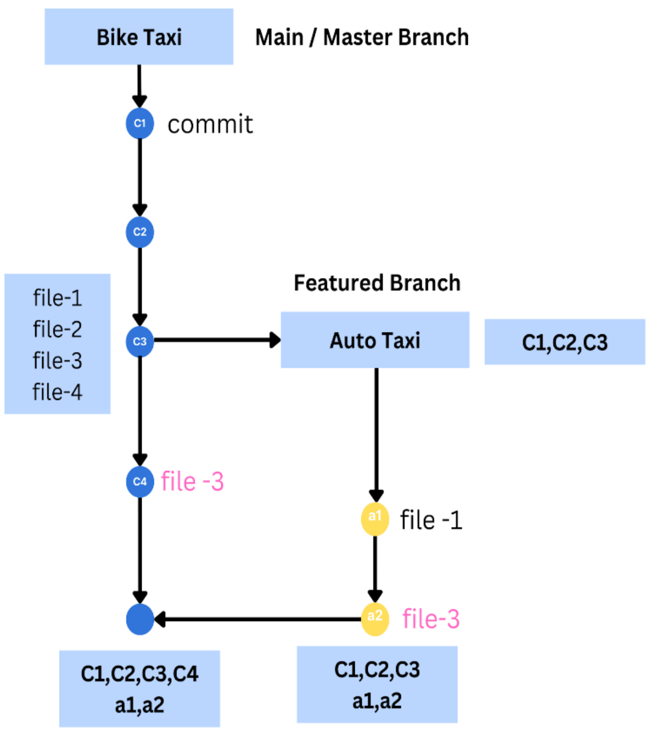
No changes are made in the parent branch while changes are made in the other branch.

## 2. Three-way Merging



Multiple branches are modified simultaneously but not same files

## 3. Merge Conflict



Same file is modified in multiple branches.

### Merging commands :

- **git merge branch-name** -> To take the changes from one branch to another.

### Undoing Changes command:

- **git reflog ->** To see the id of all the actions.
- **git checkout filename** -> Undo changes made to a file in the working directory.

- **git reset filename** -> Move a file from the staging area to the working directory.

- **git reset --soft [commit id]**-> discards the commit but changes will be present in the  staging area and working directory.

- **git reset --mixed [commit id]**-> discards commit in the local repository and discard changes in the staging area but changes will be present in the working directory.

- **git reset --hard [commit id] ->** discards commit in the local repository and discards the changes in staging area and working directory.

- **git revert filename** -> Used to create a new commit that undoes the changes made by a previous commit or a range of commits without deleting the old commit.

## To remove remote connection:

- **git remote remove origin** -> To remove the connection between remote and local repository**.**