**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.

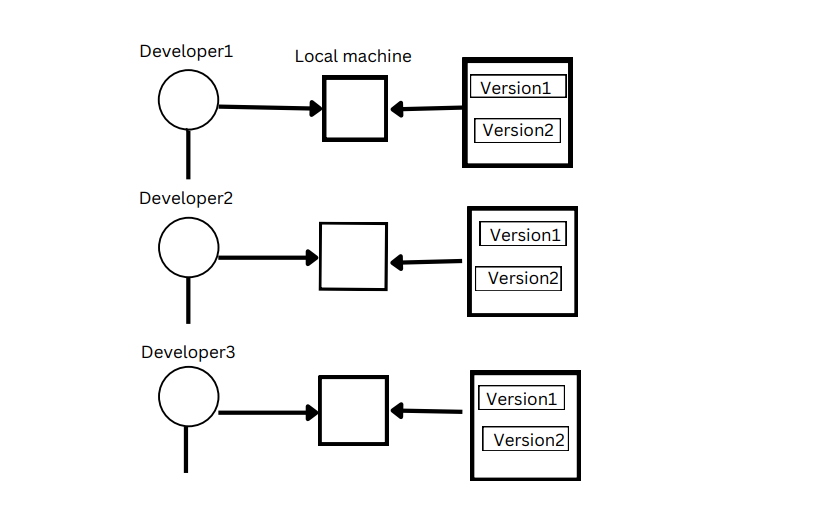
It will track (who? where? when?) all the changes made to the file.

**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.

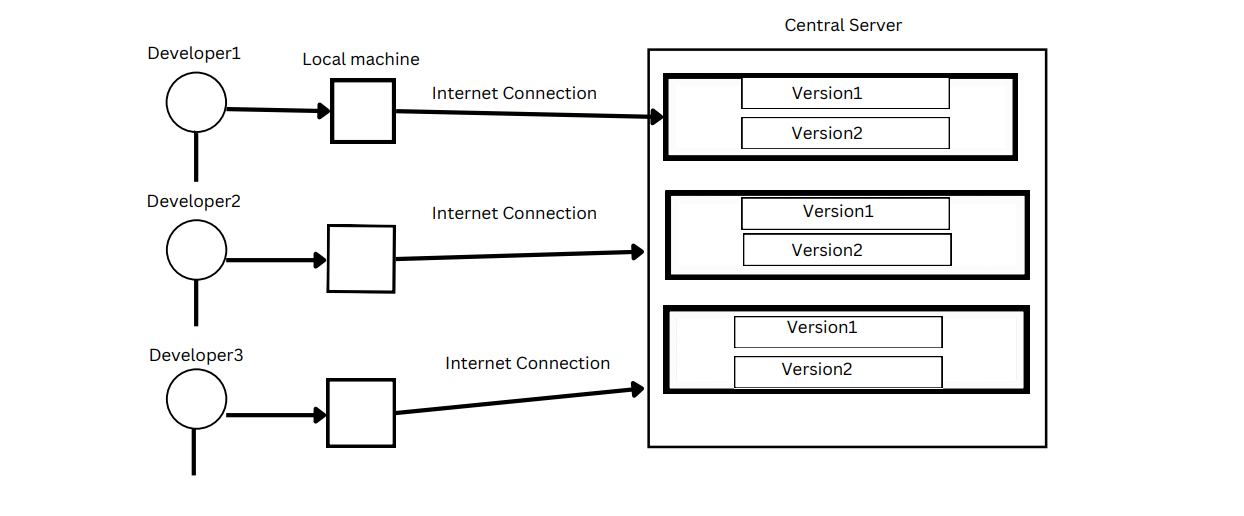
**Disadvantages of Localized VCS**

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.

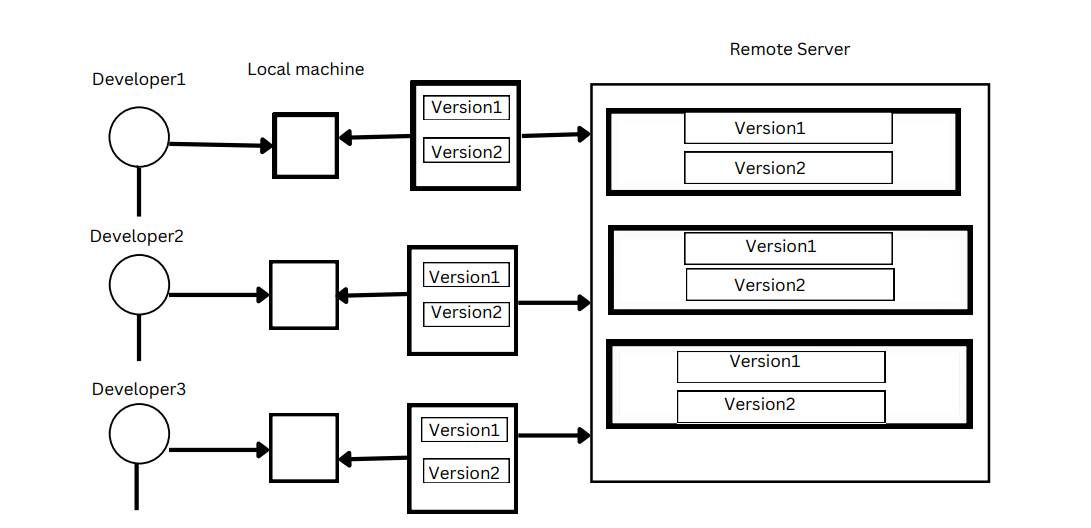
**Disadvantages**

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:** Gitbranching 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.
* Introduced for Linux Kernal to control and manage the versions of source code.
* Before GIT they were using Bitkeeper vcs tool which was paid software.

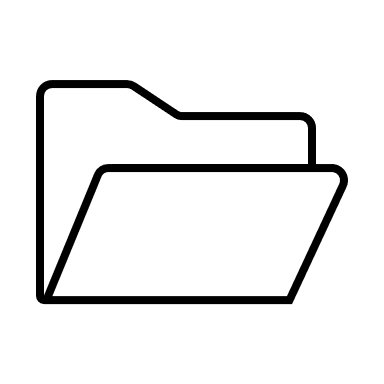
**Working Stages of Git:**

There are 3 Working Stages in GIT.

1. Working directory.

2. Staging area.

3. Local Repository.



My Project

(Folder)

**WORKING DIRECTORY**

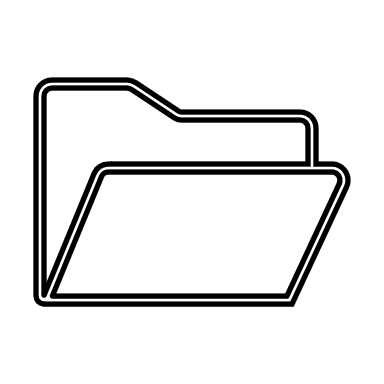
**STAGING AREA / IMAGINARY AREA**

**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:**

****

**My Project (Folder)**

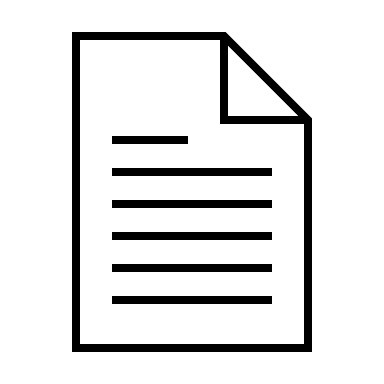
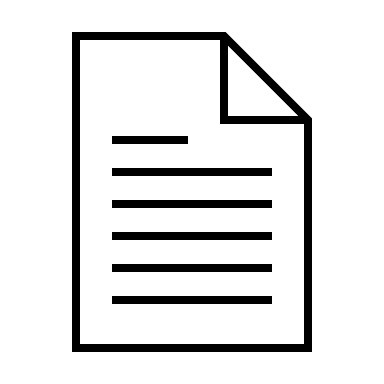
**Remote Repository**

**Local Repository**

**Staging Area/ Imaginary Area**

**Working Directory**

**A black background with a black square

Description automatically generated with medium confidence**

****

File1

File1

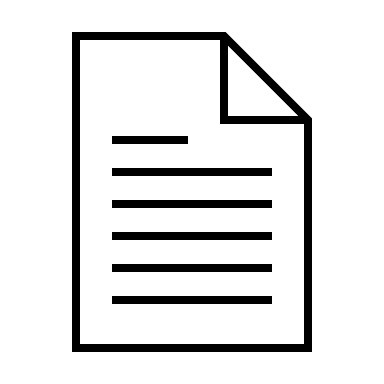
**Pushing**

**Adding**

File1

File1

**Fetch**

****

**Committing**

File2

**Pulling**

**Workflow of GIT:**

There are 7 stages in workflow:

1. Initialization
2. Configuration
3. Adding
4. Committing
5. Connection
6. Pushing
7. Pulling
8. **Initialization:** To initialize the GIT into local folder.
9. **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.
10. **Adding:** Moving files from working directory to staging area.
11. **Committing:** Moving files from staging area to local repository by committing.
12. **Connection:** Establishing connection between local repository and remote repository.
13. **Pushing:** Copying files from local repository to remote repository.
14. **Pulling:** Copying files from remote repository to working directory.

**Basic Commands to perform above operations.**

1. To initialize the local repository.

* **git init**

1. 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**](mailto:abc@gmail.com)**”**

1. Adding files from working directory to staging area.

* **git add filename** 🡪to add single file.
* **git add . 🡪** to add all the files.
* **git add \*.extension 🡪** to add only the extensions files eg., \*.java, \*.txt etc.

1. Committing files from staging area to local repository.

* **git commit -m “any message”**

1. To establish connection between local and remote repository.

* **git remote add [alias name**🡪origin**] URL** (remote URL)

1. To push files from local to remote repository.

* **git push origin master** (master is default user branch)

1. To pull the files from remote repository to local repository.

(pull is the combination of fetch and merge)

* **git pull origin master**
* **git pull remote\_url**

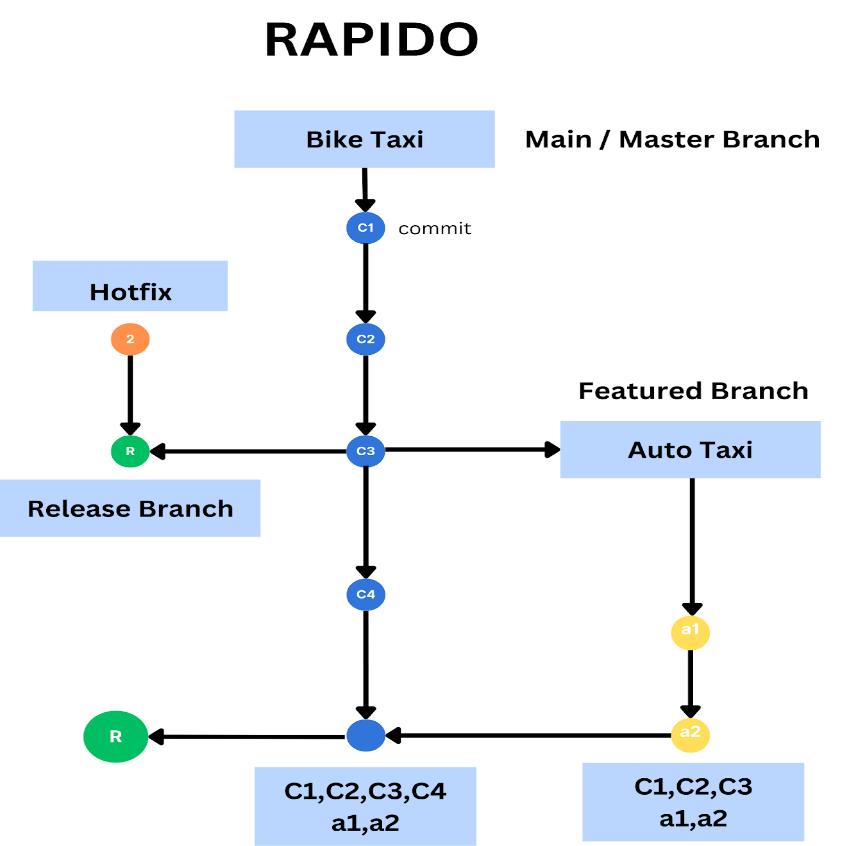
**Fetch commands:**

* **git fetch origin master** -> To fetch the files from remote repository to the local repository.
* **git checkout origin/master filename** -> To take a specific file from local repo to working directory after fetch.
* **git merge origin/master** -> To merge local repository to working directory.

**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.
  + **git rm filename**
* To remove file from remote repository.
  + **git rm - -cached filename**
* To edit email and username.
  + **git config - -edit.**
* To fetch(it will not merge files with existing files) the files from remote to local repository.
  + **git fetch origin branchname(master)**
  + **git fetch remote\_url**

**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** -> Tocreate branch

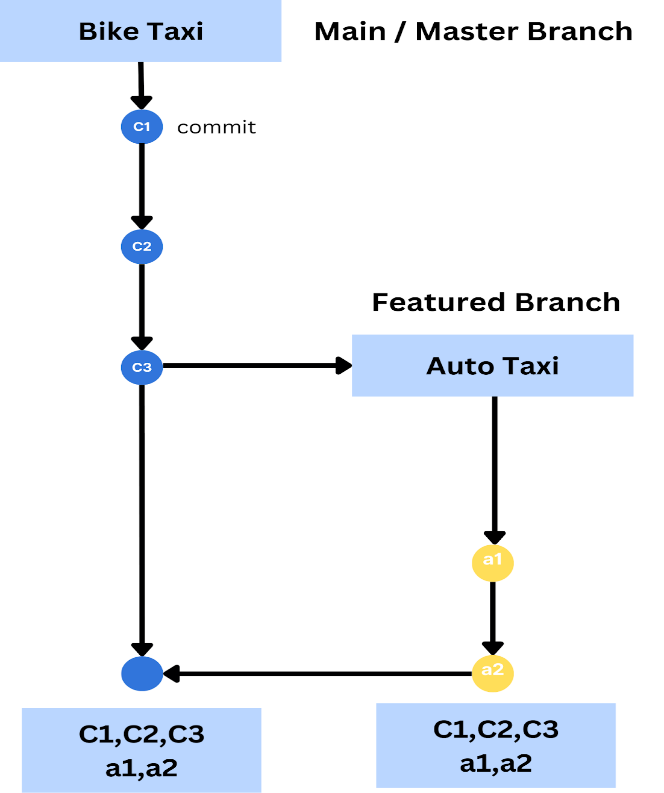
* **git switch branchname** -> To switch branch

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

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

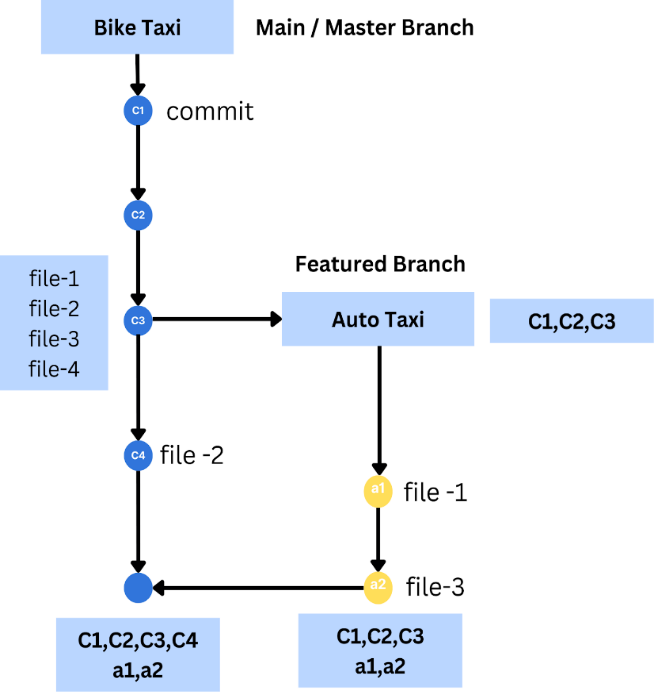
**Merging Branches**

**1. Two-way merging**

****

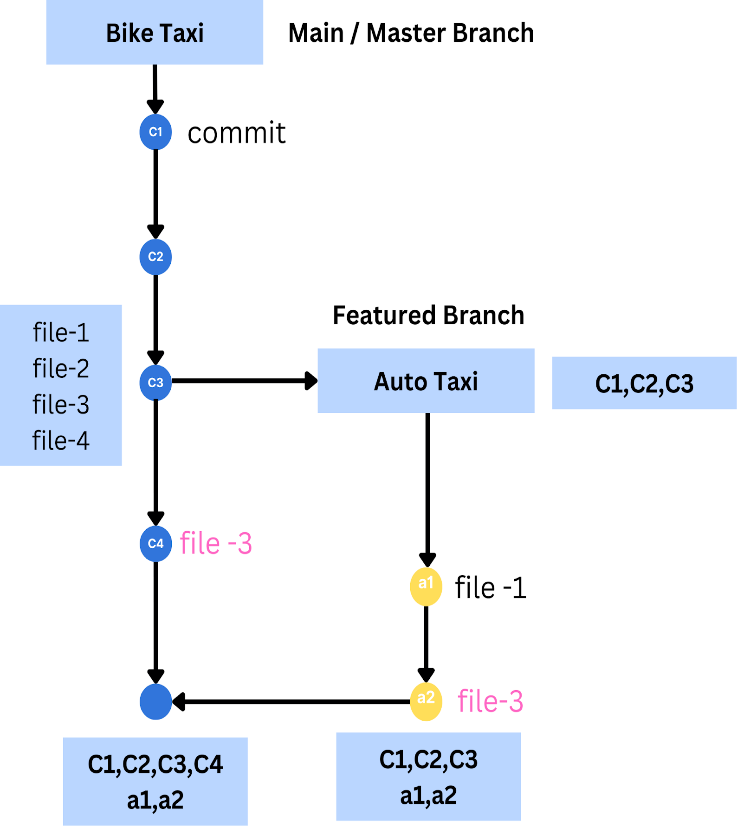
This is called as Two-way merging or Fast Forward Merging

**2. Three-way Merging**

****

This is also called as ORT(Ostensible Recursive Twin)

1. **Merge Conflict**

****

You will get the merge conflict here because You are modifying the same file-3 in the two branches hence you have to resolve the merge conflict manually.

**Merging commands :**

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

* **git rebase branchname** -> To take the changes of one branch to the tip of another branch and creates the linear history.

* **git cherry-pick commit\_hash** ->To take the specific 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 checkout .** -> Undo changes made to all the files in the working directory.
* **git reset filename** -> Move a file from the staging area to the working directory.

* **git reset .** -> Move all the files from the staging area to the working directory.
* **git reset --soft** ->discards the commit but changes will be present in the  staging area and working directory.

* **git reset --mixed** -> 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 -**>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.

**Stash Commands:**

* **git stash** -> To store the changes temporarily in working directory
* **git stash apply** -> To  undo the stash
* **git stash list ->** To see stash list
* **git stash apply stash@{0} ->** To apply specific stash
* **git stash clear ->** To clear the stash list

**Deleting the files:**

* **git rm fileanme** -> Deletes the file locally.
* **git rm --cached filename** -> Deletes the file remotely.

**Tagging commands:**

* **git tag tagname** ->To create a tag.
* **git tag** -> To see the tags.
* **git push origin tagname** -> To push a tag into remote repository.
* **git tag -d tagname** -> To delete a tag.

**To remove remote connection:**

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

**To Edit Configuration:**

* **git config --edit** -> To edit the configuration.

**GIT INTERVIEW QUESTIONS**

**1. What is Version control system?**

Managing or control over the versions of source code is called as Version control system.

**2. What is GIT, full form of GIT and why GIT is used?**

GIT is a version control tool which follows distributed version control system (DVCS).

The full form of GIT is Global Information Tracker.

GIT is used to track the changes in source code and save the versions of the source code.

**3. What is git repository**?

A git repository is a file structure where it stores all the project-based files in remote repository or local repository.

**4. What does git clone will do and how will you clone the repository?**

The git clone creates a copy (or clone) of an existing git repository. Generally, it is used to get a copy of the remote repository to the local repository.

To clone the repository execute the following command

*git clone <repository\_url>*

**5. What does git fetch will do and how will you perform fetch operation?**

The git fetch command is used to retrieve changes from a remote repository but doesn't merge those changes into your local branches.

To fetch the changes form remote repository we’ll use command as

*git fetch origin master (or)*

*git fetch URL*

**6. What does git pull will do how to get pull request from remote**

**repository**?

The git pull command combines two actions: fetching changes from remote repository and merging them into the current branch.

• It’s a combination of fetch and merge.

To get pull request execute the following command

*git pull origin master (or)*

*git pull URL*

**7. What is the difference between git fetch, git clone and git pull?**

*git* *fetch* retrieves changes from a remote repository to your local repository.

*git* *clone* copies an entire remote repository to your local machine.

*git* *pull* fetches and merges changes from a remote repository into your current branch in one step.

**8. What are the working stages in git?**

In git we have 3 working stages

1. Working directory

2. Staging area

3. Local Repository.

**9. What is the workflow of git?**

1. initialization

2. configuration

3. adding

4. committing

5. Establishing connection between local repository and remote repository

6. Pushing

7. Pulling

**10. What is branching in git?**

Branching is nothing but working on different features by diverging from main line or main branch(master) in git.

**11.What are the types of branches in git?**

1. Master branch-This is the primary branch of the repository.

2. Feature branch- Feature branches are used to develop new features or functionalities.

3. Release branch- Release branches are used to prepare a new release version of the software.

4. Hotfix branch- It is used to fix the error if there is anything in release branch.

**12. What is fast forwarding merging?**

Fast-forward merging is a type of merge operation in Git that occurs when merging two branches where one branch's commits are directly ahead of the other, forming a linear history.

**13. When does merge conflict occurs?**

If there are same file name in both master and slave in 3 way merging then there will be a merge conflict.

**14. What is the command for creating a branch?**

git branch branchname

**15**. **Which command is used to switch to a branch?**

git checkout branchname or git switch branchname

**16. How to create and switch a branch in single line ?**

git checkout -b branchname

**17. How to delete a branch in local repository?**

git branch -d branchname (or)

git branch -D branchname

**18. How to delete a branch in remote repository?**

git push origin -d branchname

**19. What is merge in git?**

Merging is a Git operation that integrates changes from one branch into another.

**20. How to undo the changes in working directory before staging a file?**

git checkout filename

**21. How to undo changes in working directory before staging all files?**

git checkout .

**22. What is the purpose of git reset?**

git reset will undo the changes from local repository, staging area and working directories.

**23. What are the different modes of git?**

Soft, mixed and hard

**24. What is the purpose of git revert?**

The git revert command in Git is used to create a new commit that undoes the changes made by a previous commit.

**25. How to delete files in git from local repository and remote repository?**

git rm filename – to delete a file in local repository

git rm -- cached filename- to delete file in remote repository.

**26. What is rebase in git?**

In Git, rebase is a command which is used to merge changes from one branch onto another branch in a more linear fashion.

**27. What is the difference between git merge and git rebase?**

git merge: It merges the branches by creating a new merge commit that combines the changes from two branches.

git rebase: It merged the branches by rewriting the commit history of the current branch.

**28.What is git cherry pick?**

git cherry pick is a command used in git which is used to pick and apply specific commits from one branch to another.

**29.What is the difference between git rebase and git cherry pick?**

git cherry pick allows you to pick individual commits and apply them onto a different branch without merging the entire branch. git rebase rewrites the commit history of the current branch by moving or reapplying commits from another branch onto it.

**30.What is git stash?**

The `git stash` command temporarily stores your working directory's modifications, allowing you to switch branches without discarding your current progress.

**31. How to view commit history?**

By using command

git log

To view commit history in oneline

git log --oneline

**32. How to stage a files from working directory?**

By using command

git add .

To add particular file for staging use git add filename

**33.How to commit the files present in staging area?**

git commit -m “any message”