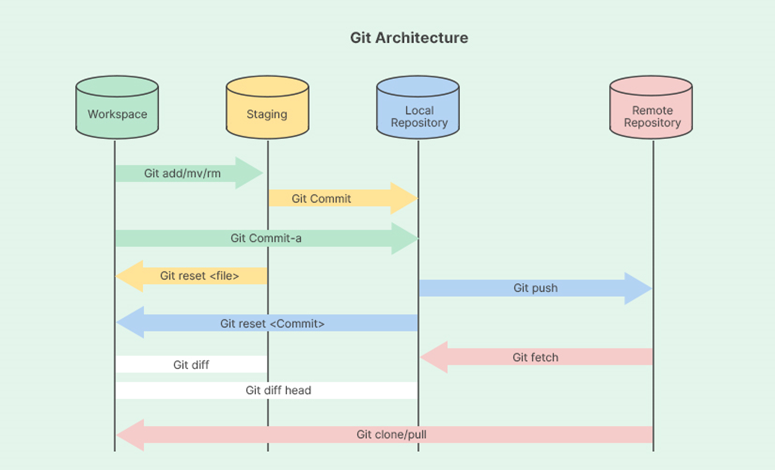
**Day-6 Assignment 1,2,3&4**

**Assignment 1:**

Explain the Architecture of Git



**Working Directory:**

* This is a local directory where we clone or create a Git repository. It contains the files we are working on.
* We can make changes to these files, and Git tracks those modifications.

**Staging Area (Index):**

* This is a temporary holding area for changes we plan to commit to the Git repository.
* We can add specific files or directories to the staging area using the git add command.
* Git only commits the changes that are staged.

**Local Repository:**

* This is a hidden directory (.git) within our working directory. It stores the Git metadata, including the commit history, references to committed file snapshots (blobs), and branch information.
* The commit history is a chronological record of all the changes made to our project, with each commit representing a specific point in time.

**Remote Repository:**

* This is a copy of the Git repository hosted on a remote server, like GitHub, GitLab, or any other hosting service.
* We can push our local commits to the remote repository to share the changes and collaborate with others.
* We can also pull changes from the remote repository to get updates made by other collaborators.

**Assignment 2:**

Explain all the git commands

**1. git init**

-> Initializes a new Git repository in the current directory.

Ex: git init

**2. git clone**

->Creates a copy of an existing repository into a new directory.

Ex: git clone <repository\_url>

**3. git add**

-> Adds file contents to the staging area.

Ex: git add <file> or git add . to add all files.

**4. git commit**

-> Records changes to the repository.

Ex: git commit -m "commit message"

**5. git status**

-> Displays the state of the working directory and staging area.

Ex: git status

**6. git log**

-> Shows the commit history for the repository.

Ex: git log

**7. git branch**

-> Lists, creates, or deletes branches.

Ex: git branch to list branches, git branch <branch\_name> to create a branch, git branch -d <branch\_name> to delete a branch.

**8. git checkout**

-> Switches branches or restores working directory files.

Ex: git checkout <branch\_name> or git checkout <commit\_id> for switching branches, git checkout -- <file> to discard changes in a file.

**9. git merge**

-> Merges branches into the current branch.

Ex: git merge <branch\_name>

**10. git remote**

-> Manages set of tracked repositories.

Ex: git remote to list remotes, git remote add <name> <url> to add a remote.

**11. git pull**

-> Fetches and integrates with another repository or a local branch.

Ex: git pull <remote> <branch>

**12. git push**

-> Updates remote refs along with associated objects.

Ex: git push <remote> <branch>

**13. git fetch**

-> Downloads objects and refs from another repository.

Ex: git fetch <remote>

**14. git diff**

->Shows changes between commits, commit and working tree, etc.

Ex: git diff or git diff <commit\_id> for specific commit differences.

**15. git stash**

-> Stashes the changes in a dirty working directory away.

Ex: git stash or git stash pop to reapply stashed changes.

**16. git rebase**

-> Reapplies commits on top of another base tip.

Ex: git rebase <branch\_name>

**17. git reset**

-> Resets current HEAD to the specified state.

Ex: git reset --hard <commit\_id> or git reset <commit\_id> for soft reset.

**18. git rm**

-> Removes files from the working tree and from the index.

Ex: git rm <file>

**19. git mv**

-> Moves or renames a file, a directory, or a symlink.

Ex: git mv <source> <destination>

**20. git tag**

-> Creates, lists, or deletes tags.

Ex: git tag to list tags, git tag <tag\_name> to create a tag, git tag -d <tag\_name> to delete a tag.

**21. git show**

-> Displays various types of objects.

Ex: git show <object>

**22. git blame**

-> Shows what revision and author last modified each line of a file.

Ex: git blame <file>

**23. git revert**

-> Creates a new commit that undoes the changes made by an earlier commit.

Ex: git revert <commit\_id>

**24. git log --oneline**

-> Shows a simplified log of commits.

Ex: git log --oneline

**25. git commit --amend**

-> Modifies the most recent commit.

Ex: git commit --amend -m "new commit message"

**26. git clean**

-> Removes untracked files from the working tree.

Ex: git clean -f (forcefully clean untracked files)

**27. git cherry-pick**

-> Applies the changes introduced by some existing commits.

Ex: git cherry-pick <commit\_id>

**28. git reflog**

-> the history of all actions on the references.

Ex: git reflog

**29. git config**

-> Gets and sets repository or global options.

Ex: git config --global user.name "Your Name" to set a global username, git config –list to list settings.

**30. git bisect**

-> Uses binary search to find the commit that introduced a bug.

Ex: git bisect start, git bisect good <commit\_id>, git bisect bad <commit\_id>

**Assignment 3:**

3. Write a steps to create a new branch and merge with master branch

**Step 1: Check out the Master Branch**

Ensure you are on the master branch before creating a new branch.

Command: git checkout master

**Step 2: Pull the Latest Changes**

Update your local master branch with the latest changes from the remote repository.

Command: git pull origin master

**Step 3: Create a New Branch**

Create a new branch from the updated master branch.

Command: git checkout -b new-branch-name

**Step 4: Make Changes in the New Branch**

Work on your changes and commit them to the new branch.

Command: git add .

Command: git commit -m "Description of the changes made"

**Step 5: Push the New Branch to the Remote Repository**

Push your new branch to the remote repository to share your changes.

Command: git push origin new-branch-name

**Step 6: Check Out the Master Branch Again**

Switch back to the master branch to prepare for merging.

Command: git checkout master

**Step 7: Pull the Latest Changes on the Master Branch**

Ensure your local master branch is up-to-date before merging.

Command: git pull origin master

**Step 8: Merge the New Branch into the Master Branch**

Merge your new branch into the master branch.

Command: git merge new-branch-name

**Step 9: Resolve Any Merge Conflicts**

If there are any merge conflicts, resolve them manually, then add and commit the resolved files.

Command: git add resolved-file.txt

Command: git commit -m "Resolved merge conflicts"

**Step 10: Push the Updated Master Branch to the Remote Repository**

Push the merged changes on the master branch to the remote repository.

Command: git push origin master.

**Assignment 4:**

4. Explain fork and git clone with examples.

**Fork:**

It is a process on Git, where you create a personal copy of someone else's repository. This is typically used when you want to make changes to a project that you don't have write access to.

**Example:**

1. Fork a repository:

○ On GitHub, navigate to the repository you want to fork.

○ Click the "Fork" button at the top right of the repository page.

2. Clone your forked repository:

○ After forking, you need to clone your forked repository to your local machine.

**Command:** git clone <https://github.com/your-username/forked-repo.git>

**Git Clone :**

It is a process of creating a local copy of a remote repository. This can be done for any repository you have access to, whether it's your own or someone else's.

Example: Clone a repository: Use the git clone command to create a local copy of the repository.

**Example:**

Clone a repository: Open your terminal or Git Bash and use the git clone command followed by the URL of the repository you want to clone. For example, to clone a repository named example-repo from GitHub:

Command: git clone https://github.com/username/example-repo.git