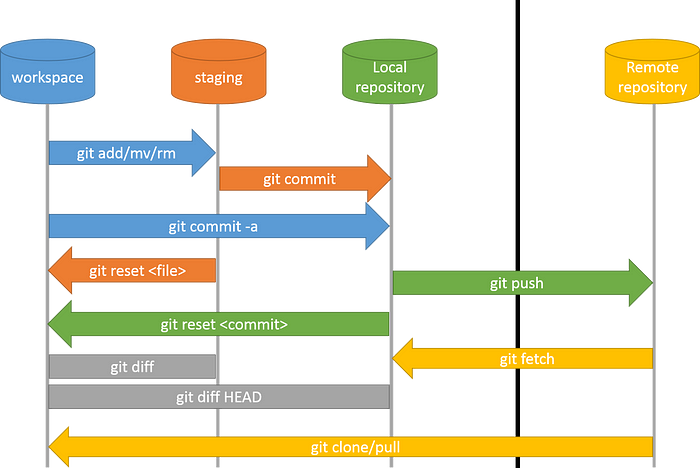
**Assignment 1: Explain the architecture of git.**



Git's architecture is designed to be flexible, scalable, and efficient for managing source code changes. It has a client-server architecture that allows multiple developers to work on the same codebase simultaneously without needing a centralized server.

Git’s architecture revolves around three key components:

1. **Working Directory**

This is where you make changes to your files. It contains the current state of your project. However, not all changes in the working directory are tracked by Git yet. It’s like a sandbox where you experiment and create new features.

1. **Staging Index (or “Index”):**

Positioned between the repository and the working directory, the staging index acts as a bridge. When you’re ready to commit changes, you add them to the index. These changes represent what you plan to commit to the repository. It’s like preparing a batch of changes for the next snapshot.

1. **Repository:**

The repository is the heart of Git. It’s where all your project’s history and versions are stored. Think of it as a time capsule that captures every change made to your files and directories. The repository tracks snapshots of your project at each commit, allowing you to revisit any version with ease.

**Assignment 2: Explain all the git commands.**

* **git init:** initializes a new Git repository.

git init <repository name>

* **git clone**: clones a repository from a remote source.

git clone <repository url>

* **git add:** adds files to the staging area.

git add .

* **git commit:** commits changes to the repository.

git commit -m "<commit message>"

* **git status:** shows the status of the repository.

git status

* **git push:** pushes changes to a remote repository.

git push <branch name>

* **git pull:** pulls changes from a remote repository.

git pull <branch name>

* **git branch:** shows a list of branches in the repository.

git branch <new branch>

* **git checkout:** switches to a different branch.

git checkout <branch name>

* **git merge:** merges changes from one branch into another.

git merge <branch name>

* **git log:** shows a log of all the commits in the repository.

git log

* **git reset:** resets the repository to a previous commit.

git reset <commit>

* **git revert:** undoes a previous commit.

git revert <commit>

* **git stash:** temporarily saves changes that are not yet ready to be committed.

git stash save <message>

* **git tag:** creates a new tag in the repository.

git tag <tag-name>

* **git remote:** shows a list of remote repositories.

git remote

* **git fetch:** fetches changes from a remote repository.

git fetch <remote>

* **git config**: sets configuration options for Git.

git config --global user.name "Sara Doe"

git config --global user.email [anu@gmail.com](mailto:anu@gmail.com)

* **git diff:** shows the differences between commits or branches.

gif diff <branch1> <branch2>

* **git blame:** shows who made changes to each line of a file.

git blame <myfile.txt>

* **git grep:** searches for text in the repository.

git grep "example"

* **git show:** used to display information about the commit with the ID. It shows the commit message, author, date, and changes made in the commit.

git show <commitID>

* **git whatchanged**: used to display the commit history along with the changes made in each commit. It shows the commit message, author, date, and file changes.

git whatchanged

* **git cherry-pick:** applies a specific commit to the current branch.

git cherry-pick commitSHA

* **git bisect:** helps identify the commit that introduced a bug.

git bisect start

git bisect good v1.0

git bisect bad HEAD

**Assignment 3. write a step to create a new branch and merge with master branch.**

1. **Create a new branch:**

* First, ensure you’re on the master branch:

git checkout master

* Create a new branch (replace <branch-name> with your desired branch name):

git checkout -b <branch-name>

1. **Make changes:**

* Switch to the new branch:

git checkout <branch-name>

* Make your changes to the code.

1. **Commit your changes:**

* Add your changes to the staging area:

git add .

* Commit the changes:

git commit -m "Your commit message here"

1. **Merge the branch with master:**

* Switch back to the master branch:

git checkout master

* Merge the changes from your branch into master:

git merge <branch-name>

1. **Resolve any conflicts:**

* If there are conflicts, resolve them manually.
* After resolving, commit the merge:

git commit -m "Merge branch '<branch-name>' into master**"**

1. **Push changes to remote:**

* Finally, push the changes to the remote repository:

git push origin master

**Assignment 4. explain fork and git clone with example.**

1. **Git Fork:**

* **Purpose:**

Forking creates a server-side copy of an existing public repository. It’s commonly used when you want to contribute to a project or create your own version of someone else’s code.

* **Workflow:**

1. You fork the original repository on a Git hosting platform (e.g., GitHub).
2. This creates a copy of the repository under your account.
3. You can pull updates from the original repository into your forked repository using git pull.
4. To contribute changes back to the original repository, you create a pull request (PR). Maintainers review your changes and decide whether to merge them.
5. How to Fork:
6. On GitHub, go to the public repository and click the “Fork” button.
7. Alternatively, use the GitHub CLI Tools:
8. $ gh repo fork <your\_repository\_here>

* **Use Cases:**

Collaborative open-source contributions.

Personal experimentation.

Creating a personal backup of a public repository.

1. **Git Clone:**

* **Purpose:**

Cloning duplicates the entire contents of an existing Git repository onto your local machine.

* **Workflow:**

1. You clone a repository to work with it locally.
2. The clone includes all branches and commit history.
3. You can make changes, create new branches, and work offline.
4. However, changes made in your clone don’t directly impact the original repository.
5. To sync your changes with the original repository, you push them back to the remote.
6. How to Clone:
7. $ git clone <repository\_url>

* **Use Cases:**

Offline access.

Individual development.

Obtaining a copy of the codebase for further use.