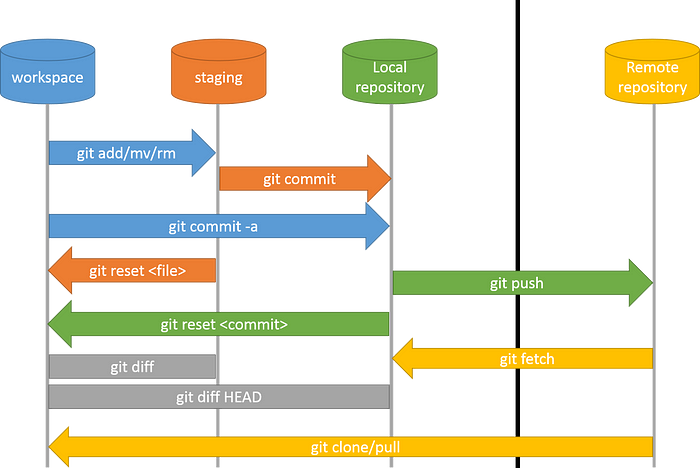
**Assignment 1: Explain the architecture of git.**

Git is a distributed version control system designed to manage source code history and facilitate collaboration among developers. Its architecture is decentralized, allowing each user to have a complete local copy of the repository, including its entire history

**Working Directory:**

* + The working directory is where files are stored and where you edit them. It represents the current state of your project and includes all files and directories in their latest form.

1. **Staging Area (Index):**
   * The staging area is a crucial concept in Git. It acts as a middle ground between your working directory and the repository. Files that are staged are ready to be committed to the repository. This allows you to selectively choose which changes to include in the next commit.
2. **Local Repository:**
   * The local repository is where Git stores all the committed changes and their history on your local machine. It contains the complete history of the project, including all branches and commits.
3. **Remote Repository:**
   * Git also supports remote repositories, which are hosted on a server (like GitHub, GitLab, or Bitbucket) and act as centralized hubs where developers can collaborate. Remote repositories allow multiple developers to work on the same project simultaneously and facilitate sharing of code changes.
4. **Commit Objects:**
   * Git stores data as snapshots of the project’s directory at each commit. Each commit is represented by a commit object that contains metadata (like author, timestamp, and commit message) and references to the parent commit(s).
5. **Branches:**
   * Branches in Git are lightweight pointers to specific commits. They allow for parallel development by enabling developers to work on separate features or fixes independently. Branches can be created, switched between, merged, and deleted as needed.



**Assignment 2: Explain 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 "ankit19"

git config --global user.password ankit1234

* **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

Certainly! Here’s an expanded and revised version with added points to differentiate it further:

**Assignment 4: Understanding Fork and Git Clone with Examples**

**1. Git Fork:**

**Purpose:**

Forking creates a server-side copy of an existing public repository, allowing you to freely experiment or contribute without affecting the original repository.

**Workflow:**

* You fork the original repository on a Git hosting platform like GitHub, creating a duplicate under your account.
* Your forked repository remains independent but can pull updates from the original using git pull.
* Contributions to the original repository are managed through pull requests (PRs), enabling collaboration and code review.

**Use Cases:**

* Open-source collaboration.
* Experimenting with code modifications.
* Maintaining a personal backup of important projects.

**2. Git Clone:**

**Purpose:**

Cloning allows you to create a local copy of a Git repository, providing an isolated environment for development and exploration.

**Workflow:**

* Clone a repository to your local machine to work offline and independently.
* Includes full history and branches of the remote repository.
* Changes made locally are managed through commits and can be synchronized with the remote repository via git push.

**Steps to Clone:**

$ git clone <repository\_url>

**Use Cases:**

* Exploring project code without internet access.
* Implementing new features or fixing bugs locally before pushing changes.
* Setting up a development environment with specific configurations.