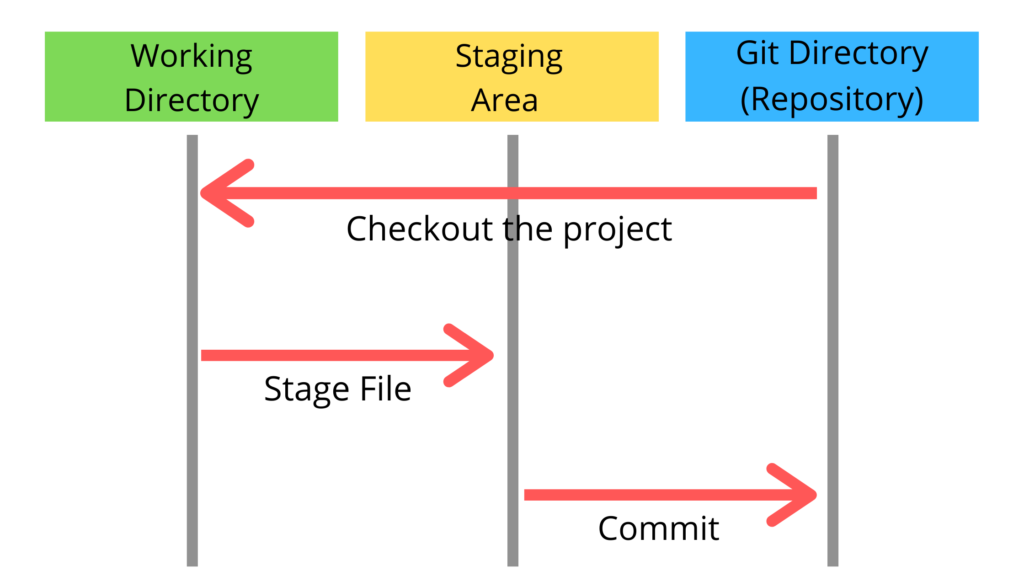
**Day-6 (Assignment-1)**

# Q)explain git architecture

Git's architecture is designed to manage and track changes to files over time, facilitating collaboration among developers and ensuring the integrity of project versions.



* Here’s an overview of Git's architecture and its key components:

### 1. **Repository (Repo)**

A Git repository is essentially a directory that contains all the files and directories for a project, along with metadata about the project's history. There are two types of repositories:

* **Local Repository**: Located on your local machine where you work with Git commands directly.
* **Remote Repository**: Hosted on a remote server (like GitHub, GitLab, Bitbucket) where collaborators can push changes and synchronize their work.

### 2. **Working Directory**

The working directory is where you edit files, and it represents the current state of the project files on your local machine. These files can be in various states:

* **Tracked files**: Files that are part of the Git repository and are tracked for changes.
* **Untracked files**: Files that exist in the working directory but are not yet tracked by Git.
* **Ignored files**: Files that are explicitly ignored and not tracked by Git (defined in .gitignore).

### 3. **Index (Staging Area)**

The index acts as a staging area between the working directory and the repository. It holds snapshots of files that are to be included in the next commit. Before committing changes, files are added to the index using git add command.

### 4. **Object Database**

Git uses a data structure called the **object database** to store all the data in the repository. This includes:

* **Blob (Binary Large Object)**: Represents the content of files. Each version of a file is stored as a blob.
* **Tree**: Represents a directory or a snapshot of the project’s state at a certain point in time. It contains pointers to blobs and other trees.
* **Commit**: A commit object contains metadata like author, commit message, and a pointer to the tree object representing the project’s state at that commit.
* **Tag**: A tag object points to a specific commit and is used to mark significant points in history, such as releases.

### 5**. Branches**

Branches in Git are lightweight movable pointers to commits. They allow multiple lines of development to coexist independently within one repository. The default branch is typically named main or master.

### 6. **HEAD**

HEAD is a reference to the current branch or commit you are working on. It points to the latest commit in the currently checked-out branch.

### 7. **Operations and Commands**

Git provides a set of commands to interact with the repository and manage its history:

* **git init**: Initializes a new Git repository.
* **git clone**: Copies a remote repository to create a local repository.
* **git add**: Adds files from the working directory to the index (staging area).
* **git commit**: Records changes to the repository with a commit message.
* **git push**: Sends local changes to the remote repository.
* **git pull**: Fetches changes from the remote repository and merges them into the local repository.
* **git merge**: Combines changes from different branches into the current branch.
* **git checkout**: Switches branches or restores working tree files.
* **git log**: Displays the commit history.

### 8. **Distributed Version Control**

Git is a distributed version control system, meaning that each developer has a complete copy of the repository, including its full history. This decentralization allows for offline work, local branching, and easier collaboration.