**Git Documentation**

Git is a version control system used for tracking changes in computer files. It is generally used for source code management in software development. Git is used to tracking changes in the source code. The distributed version control tool is used for source code management.

**Version Control System (VCS)**

VCS stands for Version Control System. It's a system that tracks changes to a file or set of files over time. VCSs automate the process of version control so you don't have to manage file versions manually.

The types of VCS are:

* Local Version Control System
* Centralized Version Control System
* Distributed Version Control System

**Local Version Control System**

A local version control system is a local database located on your local computer, in which every file change is stored as a patch. Every patch set contains only the changes made to the file since its last version. In order to see what the file looked like at any given moment; it is necessary to add up all the relevant patches to the file in order until that given moment.

The main problem with this is that everything is stored locally. If anything were to happen to the local database, all the patches would be lost. If anything were to happen to a single version, all the changes made after that version would be lost.

**Centralized Version Control System**

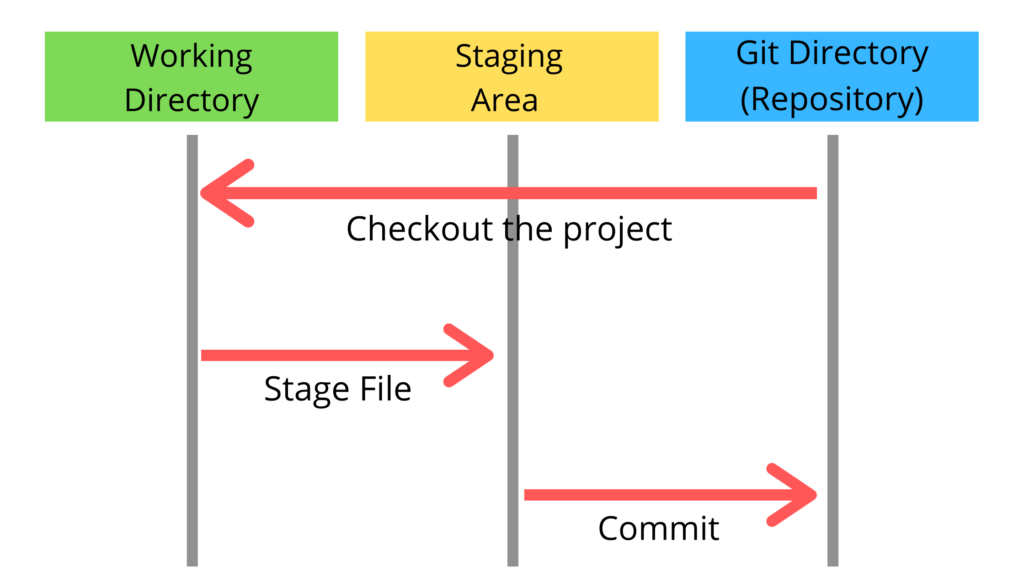
A centralized version control system has a single server that contains all the file versions. This enables multiple clients to simultaneously access files on the server, pull them to their local computer or push them onto the server from their local computer. This way, everyone usually knows what everyone else on the project is doing. Administrators have control over who can do what.

**Distributed Version Control System**

With distributed version control systems, clients don’t just check out the latest snapshot of the files from the server, they fully mirror the repository, including its full history. Thus, everyone collaborating on a project owns a local copy of the whole project, i.e., owns their own local database with their own complete history. With this model, if the server becomes unavailable or dies, any of the client repositories can send a copy of the project's version to any other client or back onto the server when it becomes available. It is enough that one client contains a correct copy which can then easily be further distributed.

**Git three-stage architecture**

Many VCS’s use a two-tier architecture i.e., a repository and a working copy. Git uses three-tier architecture i.e., a working directory, staging area and local repository. The three stages of git can store different (or the same) states of the same code in each stage.



Git – Three Stage Architecture

As we can see in above diagram there are three stages in git. When we give any file to the git that file goes from each stage at least once. The three stages of git can store different (or the same) states of the same code in each stage.

In above diagram there are three stages those are Working directory, Staging area and Git directory (Repository).

Working directory specifies the file explorer’s folder where your files are stored, staging area is an area where those files are present which you want to send to commit (to create snapshot of files), After commit is fired, files which are in staging area will move to Git Repository.

Now if you made any changes in the files which are in Git repository, those files (with changes) will be in Upstaging area. You again have to add them into Staging area and commit.

**Git commands:**

**git init:**

Initializes a new Git repository in the current directory, creating a hidden.git folder to store configuration and version history.

**git clone:**

Creates a copy of a remote Git repository on your local machine, enabling you to work on it locally.

**git add or git add.:**

Stages changes for the next commit. You can specify individual files or use . to stage all changes in the current directory.

**git commit -m "message":**

Records the staged changes into a commit with a descriptive message. This snapshot represents a point in your project's history.

**git status:**

Shows the status of your working directory, indicating which files are untracked, modified, or staged.

**git diff:**

Displays the differences between the working directory and the last committed version.

**git log:**

Shows a chronological list of commits in the current branch, including commit messages, authors, and timestamps.

**git branch:**

Lists all local branches in the repository, highlighting the current branch with an asterisk.

**git branch <branch\_name>:**

Creates a new branch with the specified name based on the current branch.

**git checkout <branch\_name>:**

Switches to the specified branch, updating your working directory to match the branch's state.

**git merge <branch\_name>:**

Combines changes from the specified branch into the current branch. Creates a merge commit if necessary.

**git pull:**

Fetches changes from a remote repository and merges them into the current branch. Equivalent to git fetch followed by git merge.

**git push:**

Uploads local commits to a remote repository, keeping both repositories in sync.

**git remote -v:**

Lists the remote repositories associated with your local repository and their URLs.

**git fetch:**

Downloads changes from a remote repository, updating your local references, but does not automatically merge them into your current branch.

**git reset <commit>:**

Moves the current branch pointer to a specified commit, effectively "rewinding" your project's history. Use with caution.

**git revert <commit>:**

Creates a new commit that undoes the changes introduced by a specified commit, preserving the commit history.

**git stash:**

Temporarily saves changes in your working directory to a "stash" so you can switch branches or perform other operations. Useful when you're not ready to commit changes.

**git tag <tag\_name>:**

Creates a lightweight tag at the current commit, allowing you to mark specific points in your project's history.