Version Control

- Version control systems, also known as source control, source code management systems, or revision control systems, are a mechanism for keeping multiple versions of your files, so that when you modify a file you can still access the previous revisions.
- Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later.
- **Version Control System (VCS)** is a software that helps software developers to work together and maintain a complete history of their work.
- Benefits of the Version Control System
- The Version Control System is very helpful and beneficial in software development; developing software without using version control is unsafe. It provides backups for uncertainty. Version control systems offer a speedy interface to developers. It also allows software teams to preserve efficiency and agility according to the team scales to include more developers.
- Some key benefits of having a version control system are as follows.
- Complete change history of the file
- Simultaneously working
- Branching and merging
- Traceability
- Types of the version control systems
  1. Local Version Control System
  2. Centralized Version Control System
  3. Distributed Version Control System

Git

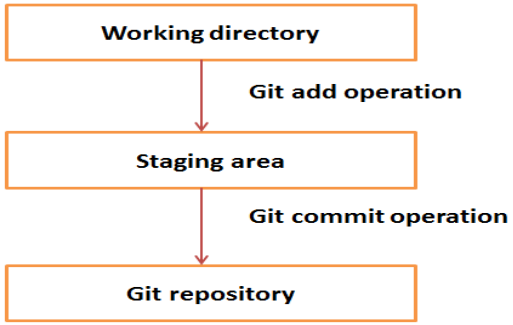
- Git is a modern and widely used **distributed version control** system in the world. It is developed to manage projects with high speed and efficiency. The version control system allows us to monitor and work together with our team members at the same workspace.
- What is Git?
- **Git** is an **open-source distributed version control system**. It is designed to handle minor to major projects with high speed and efficiency. It is developed to co-ordinate the work among the developers. The version control allows us to track and work together with our team members at the same workspace.
- Git is foundation of many services like **GitHub** and **GitLab**, but we can use Git without using any other Git services. Git can be used **privately** and **publicly**.
- Git was created by **Linus Torvalds** in **2005** to develop Linux Kernel. It is also used as an important distributed version-control tool for **the DevOps**.
- Git is easy to learn, and has fast performance. It is superior to other SCM tools like Subversion, CVS, Perforce, and ClearCase.

Git

- **Advantages of Git**
- Free and open source
  - Git is released under GPL's open source license. It is available freely over the internet. You can use Git to manage property projects without paying a single penny. As it is an open source, you can download its source code and also perform changes according to your requirements.
- Fast and small
  - As most of the operations are performed locally, it gives a huge benefit in terms of speed. Git does not rely on the central server; that is why, there is no need to interact with the remote server for every operation. The core part of Git is written in C, which avoids runtime overheads associated with other high-level languages. Though Git mirrors entire repository, the size of the data on the client side is small. This illustrates the efficiency of Git at compressing and storing data on the client side.
- Implicit backup
  - The chances of losing data are very rare when there are multiple copies of it. Data present on any client side mirrors the repository, hence it can be used in the event of a crash or disk corruption.
- Security
  - Git uses a common cryptographic hash function called secure hash function (SHA1), to name and identify objects within its database. Every file and commit is check-summed and retrieved by its checksum at the time of checkout. It implies that, it is impossible to change file, date, and commit message and any other data from the Git database without knowing Git.
- No need of powerful hardware
  - In case of CVCS, the central server needs to be powerful enough to serve requests of the entire team. For smaller teams, it is not an issue, but as the team size grows, the hardware limitations of the server can be a performance bottleneck. In case of DVCS, developers don't interact with the server unless they need to push or pull changes. All the heavy lifting happens on the client side, so the server hardware can be very simple indeed.
- Easier branching
  - CVCS uses cheap copy mechanism, If we create a new branch, it will copy all the codes to the new branch, so it is time-consuming and not efficient. Also, deletion and merging of branches in CVCS is complicated and time-consuming. But branch management with

Git Terminologies

- **Local Repository**
- Every VCS tool provides a private workplace as a working copy. Developers make changes in their private workplace and after commit, these changes become a part of the repository. Git takes it one step further by providing them a private copy of the whole repository. Users can perform many operations with this repository such as add file, remove file, rename file, move file, commit changes, and many more.
- **Working Directory**
- The working directory is the place where files are checked out. In other CVCS, developers generally make modifications and commit their changes directly to the repository. But Git uses a different strategy. Git doesn't track each and every modified file. Whenever you do commit an operation, Git looks for the files present in the staging area. Only those files present in the staging area are considered for commit and not all the modified files.
- Let us see the basic workflow of Git.
- **Step 1** – You modify a file from the working directory.
- **Step 2** – You add these files to the staging area.
- **Step 3** – You perform commit operation that moves the files from the staging area. After push operation, it stores the changes permanently to the Git repository.



Git Terminologies

- **Blobs**
- Blob stands for **B**inary **L**arge **Ob**ject. Each version of a file is represented by blob. A blob holds the file data but doesn't contain any metadata about the file. It is a binary file, and in Git database, it is named as SHA1 hash of that file. In Git, files are not addressed by names. Everything is content-addressed.
- **Trees**
- Tree is an object, which represents a directory. It holds blobs as well as other sub-directories. A tree is a binary file that stores references to blobs and trees which are also named as **SHA1** hash of the tree object.
- **Commits**
- Commit holds the current state of the repository. A commit is also named by **SHA1** hash. You can consider a commit object as a node of the linked list. Every commit object has a pointer to the parent commit object. From a given commit, you can traverse back by looking at the parent pointer to view the history of the commit. If a commit has multiple parent commits, then that particular commit has been created by merging two branches.
- **Branches**
- Branches are used to create another line of development. By default, Git has a master branch, which is same as trunk in Subversion. Usually, a branch is created to work on a new feature. Once the feature is completed, it is merged back with the master branch and we delete the branch. Every branch is referenced by HEAD, which points to the latest commit in the branch. Whenever you make a commit, HEAD is updated with the latest commit.
- **Tags**
- Tag assigns a meaningful name with a specific version in the repository. Tags are very similar to branches, but the difference is that tags are immutable. It means, tag is a branch, which nobody intends to modify. Once a tag is created for a particular commit, even if you create a new commit, it will not be updated. Usually, developers create tags for product releases.

Git Terminologies

- Clone
- Clone operation creates the instance of the repository. Clone operation not only checks out the working copy, but it also mirrors the complete repository. Users can perform many operations with this local repository. The only time networking gets involved is when the repository instances are being synchronized.
- Pull
- Pull operation copies the changes from a remote repository instance to a local one. The pull operation is used for synchronization between two repository instances. This is same as the update operation in Subversion.
- Push
- Push operation copies changes from a local repository instance to a remote one. This is used to store the changes permanently into the Git repository. This is same as the commit operation in Subversion.
- HEAD
- HEAD is a pointer, which always points to the latest commit in the branch. Whenever you make a commit, HEAD is updated with the latest commit. The heads of the branches are stored in **.git/refs/heads/** directory.
- Revision
- Revision represents the version of the source code. Revisions in Git are represented by commits. These commits are identified by **SHA1** secure hashes.
- URL
- URL represents the location of the Git repository.

Git vs GitHub/GitLab

Git	GitHub/GitLab
Git is a distributed version control tool that can manage a programmer's source code history.	GitHub is a cloud-based tool developed around the Git tool.
A developer installs Git tool locally.	GitHub is an online service to store code and push from the computer running the Git tool.
Git focused on version control and code sharing.	GitHub focused on centralized source code hosting.
It is a command-line tool.	It is administered through the web.
It facilitates with a desktop interface called Git Gui.	It also facilitates with a desktop interface called GitHub Gui.
Git does not provide any user management feature.	GitHub has a built-in user management feature.
It has minimal tool configuration feature.	It has a market place for tool configuration.

Git Commands

Task	Git commands
Create a **new local directory:**	git init
Add a specific file to staging (Git) or after a new file is created	git add <filename>
Add all changes to staging (Git) or all new files	git add –all or git add .
Commit changes locally:	git commit -m '<message>'
Connect your local repository to a remote server:	git remote add origin <server_URL>
Push changes to your remote repository:	git push <remote_name> <branch_name> usually: git push origin master

Git Commands

Task	Git commands
Copy a remote repository to your local system:	git clone <URL_to_repository>
List the **status** of the files you've changed and those you still need to add or commit:	git status
Create a new **branch**	git checkout -b <branch_name>
Switch from one branch to another:	git checkout <branch_name>
List all the branches/bookmarks in your repo with an indication of the one you are on:	git branch
Delete the feature **b**ranch	git branch -d <branch_name>

Git Commands

Task	Git commands
Push the branch/bookmark to your remote repository:	git push origin <branch_name>
Fetch and **merge** changes on the remote server to your working directory:	git pull or git pull origin <branch_name>
Merge two different revisions into one:	git merge
Show all changes made since the last commit:	git diff