**Version Control System**

**Version Control System** is software that helps software developers to work together and maintain a complete history of their work.

* Allows developers to work simultaneously.
* Does not allow overwriting each other’s changes.
* Maintains a history of every version.

**Types of Version Control Systems**

* Centralized version control system (CVCS).
* Distributed/Decentralized version control system (DVCS).

**Centralized version control system (CVCS)**

Central server to store all files and enables team collaboration. But the major drawback of CVCS is its single point of failure, i.e., failure of the central server.

**Distributed Version Control System**

DVCS clients not only check out the latest snapshot of the directory but they also fully mirror the repository. If the sever goes down, then the repository from any client can be copied back to the server to restore it. Every checkout is a full backup of the repository.

Git does not rely on the central server and that is why you can perform many operations when you are offline.

**Advantages of Git**

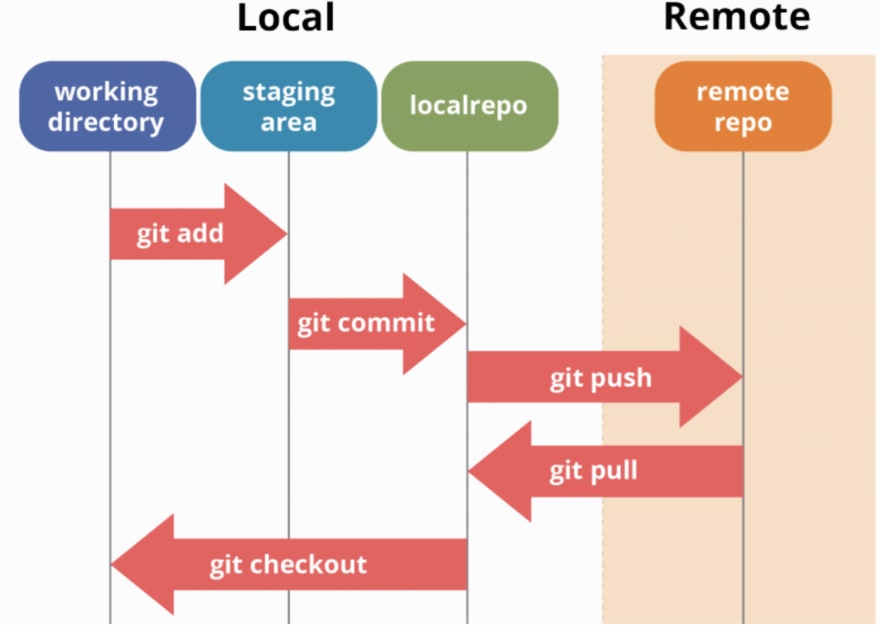
* Free and Open source.
* Fast and small
* Implicit backup
* Security
* No need of powerful hardware
* Easier branching

**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.

**Work Flow of Git**

1. You modify a file from the working directory.
2. You add these files to the staging area.
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 Installation**

Please refer below link for installing git

<https://phoenixnap.com/kb/how-to-install-git-windows#:~:text=Steps%20For%20Installing%20Git%20for%20Windows%201%20Extract,which%20extra%20option%20you%20would%20like%20to%20enable>.

**Git Commands**

**git init**

The git init command initializes a new, empty repository.

Executing git init creates a .git subdirectory in the current working directory, which contains all of the necessary Git metadata for the new repository. This metadata includes subdirectories for objects, refs, and template files. A HEAD file is also created which points to the currently checked out commit.

Compared to SVN, the git init command is an incredibly easy way to create new version-controlled projects. Git doesn’t require you to create a repository, import files, and check out a working copy. Additionally, Git does not require any pre-existing server or admin privileges. All you have to do is cd into your project subdirectory and run git init, and you'll have a fully functional Git repository.

git init <directory>

**Git Files Directory**

How the directory structure is organized and where the components of the Git tree fall in to it.

* The .**git** directory holds the meat of your local repo. It holds subdirectories for objects, refs, config, index, HEAD, and more.
* The **objects** folder is where the three objects: commit, tree and blob are housed. Each object gets its own sub folder.
  + **Blob**

Blobs, on the other hand, are just contents — binary streams of data, every blob in git are identified by its SHA-1 hash. SHA-1 hashes consist of 20 bytes

* + **Tree**

A tree is a simple object that has a bunch of pointers to blobs and other trees - it generally represents the contents of a directory or subdirectory.

* + **Commit**

Commit set a message about the changes you were done. The commit also saves a revision of the code and you can revert the code to any version anytime.

* The **refs** folder holds references to all of the objects in the directory. Let's say you have some orphaned objects on branches that no longer exist. You can go into this folder and prune them off of your branch.
* The **config** folder contains all of your configurations including the credentials you use to connect to GitHub; your email and name that you used to configure your global config.
* The **HEAD** file is what contains the pointer or reference to your working directory or branch and its latest commit.
* The **index** file is for when you use git add to stage your files for a commit. They leave the index file after the commit.

**git add**

The **git add** command adds a file to the Git staging area. This area contains a list of all the files you have recently changed.

**git commit**

The **git commit** the staged snapshot.

**git commit –a**

Commita snapshot of all changes in the working directory.

**git commit –m**

A shortcut command that immediately creates a commit with a passed commit message.

**git commit -am "commit message"**

A power user shortcut command that combines the -a and -m options. This combination immediately creates a commit of all the staged changes and takes an inline commit message.

**git commit –amend**

This option adds another level of functionality to the commit command. Passing this option will modify the last commit. Instead of creating a new commit, staged changes will be added to the previous commit.

**git status**

The git status command displays the state of the working directory and the staging area.

**git log**

The git log command displays committed snapshots.

Branching