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.

Features of Git

Some remarkable features of Git are as follows:

- **Open Source**

  Git is an **open-source tool**. It is released under the **GPL** (General Public License) license.

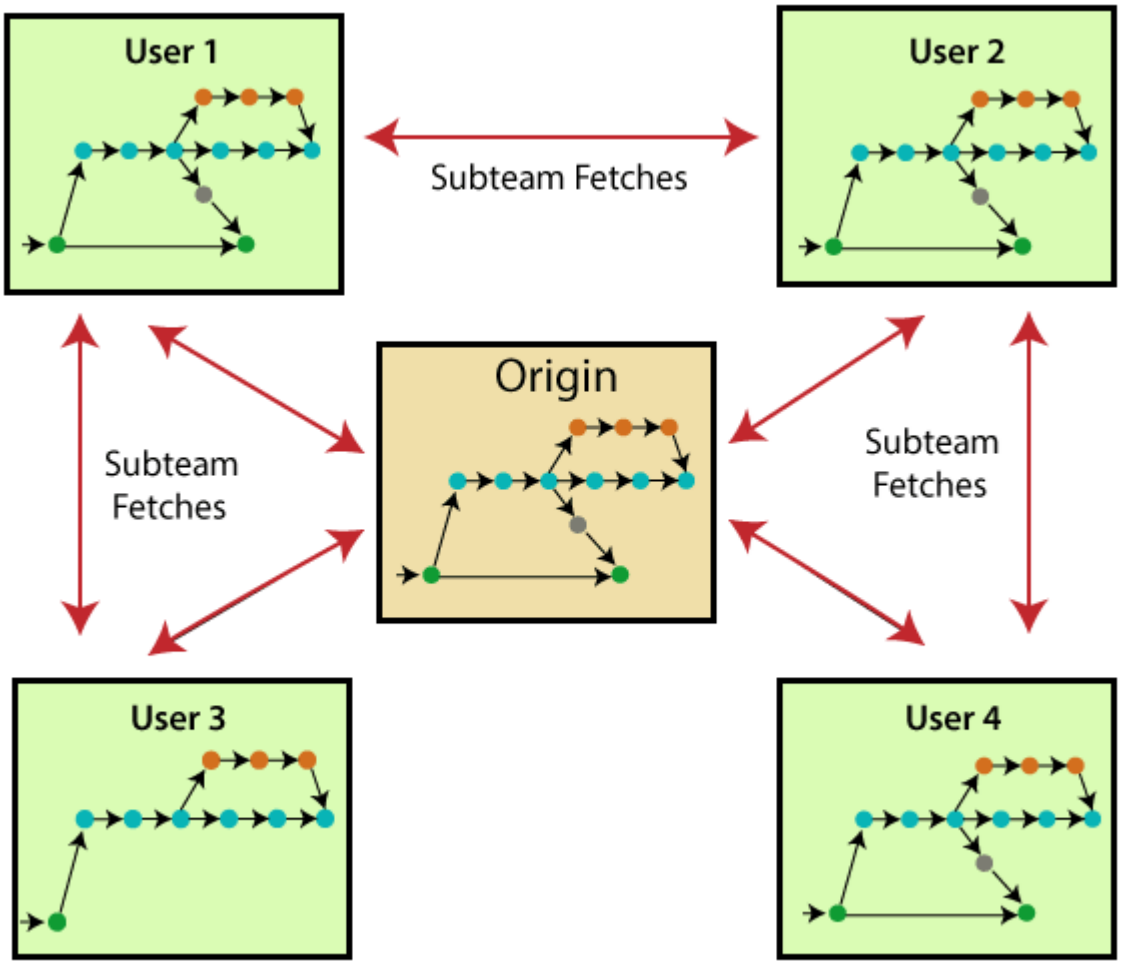
- **Scalable**

  Git is **scalable**, which means when the number of users increases, the Git can easily handle such situations.

- **Distributed**

  One of Git's great features is that it is **distributed**. Distributed means

that instead of switching the project to another machine, we can create a "clone" of the entire repository. Also, instead of just having one central repository that you send changes to, every user has their own repository that contains the entire commit history of the project. We do not need to connect to the remote repository; the change is just stored on our local repository. If necessary, we can push these changes to a remote repository.



- **Security**

  Git is secure. It uses the **SHA1 (Secure Hash Function)** to name

and identify objects within its repository. Files and commits are checked and retrieved by its checksum at the time of checkout. It stores its history in such a way that the ID of particular commits depends upon the complete development history leading up to that commit. Once it is published, one cannot make changes to its old version.

- **Speed**

  Git is very **fast**, so it can complete all the tasks in a while. Most of the git operations are done on the local repository, so it provides a **huge speed**. Also, a centralized version control system continually communicates with a server somewhere. Performance tests conducted by Mozilla showed that it was **extremely fast compared to other VCSs**. Fetching version history from a locally stored repository is much faster than fetching it from the remote server. The **core part of Git** is **written in C**, which **ignores** runtime overheads associated with other high-level languages. Git was developed to work on the Linux kernel; therefore, it is **capable** enough to **handle large repositories** effectively. From the beginning, **speed** and **performance** have been Git's primary goals.

- **Supports non-linear development**

  Git supports **seamless branching and merging**, which helps in visualizing and navigating a non-linear development. A branch in Git represents a single commit. We can construct the full branch structure with the help of its parental commit.

- **Branching and Merging**

  Branching and merging are the **great feature**s of Git, which makes it different from the other SCM tools. Git allows the **creation of multiple branches** without affecting each other. We can perform tasks like **creation**, **deletion**, and **merging** on branches, and these tasks take a few seconds only. Below are some features that can be achieved by branching:

  - We can **create a separate branch** for a new module of the project, commit and delete it whenever we want.
  - We can have a **production branch**, which always has what goes into production and can be merged for testing in the test branch.

- We can create a **demo branch** for the experiment and check if it is working. We can also remove it if needed.
- The core benefit of branching is if we want to push something to a remote repository, we do not have to push all of our branches. We can select a few of our branches, or all of them together.
-

- **Data Assurance**

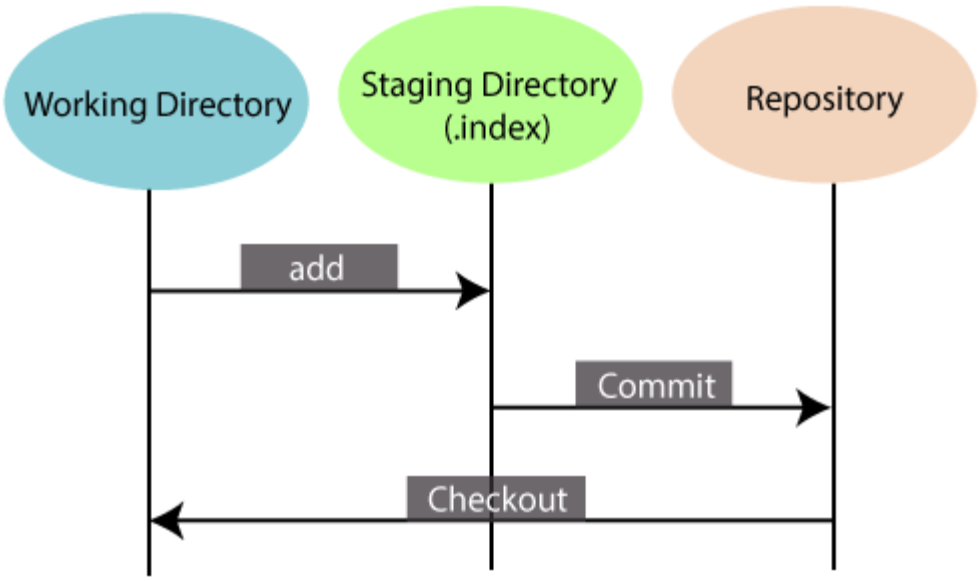
The Git data model ensures the **cryptographic integrity** of every unit of our project. It provides a **unique commit ID** to every commit through a **SHA algorithm**. We can **retrieve** and **update** the commit by commit ID. Most of the centralized version control systems do not provide such integrity by default.

- **Staging Area**

The **Staging area** is also a **unique functionality** of Git. It can be considered as a **preview of our next commit**, moreover, an **intermediate area** where commits can be formatted and reviewed before completion. When you make a commit, Git takes changes that are in the staging area and make them as a new commit. We are allowed to add and remove changes from the staging area. The

staging area can be considered as a place where Git stores the changes.

Although, Git doesn't have a dedicated staging directory where it can store some objects representing file changes (blobs). Instead of this, it uses a file called index.



Another feature of Git that makes it apart from other SCM tools is that **it is possible to quickly stage some of our files and commit them without committing other modified files in our working directory.**

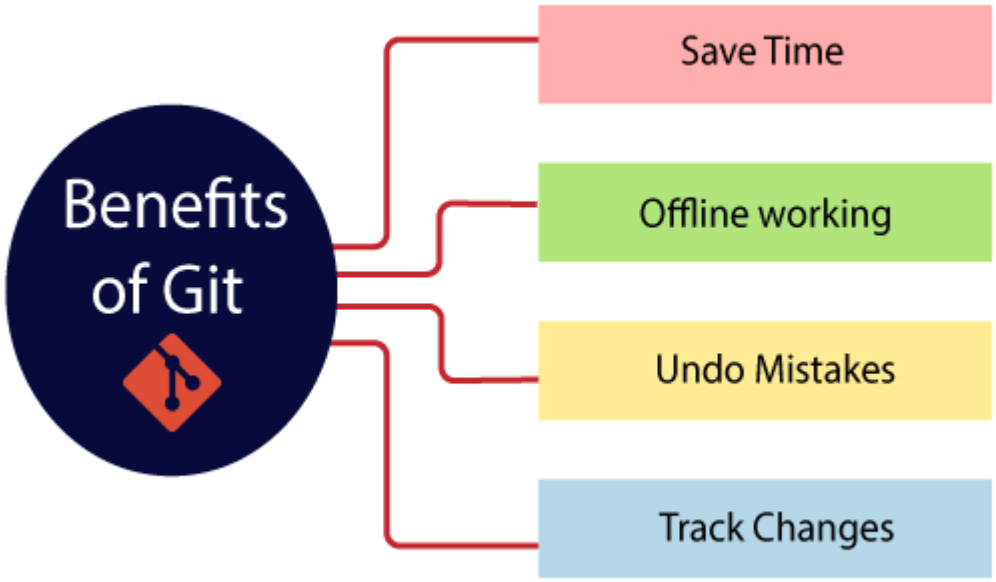
- **Maintain the clean history**

Git facilitates with Git Rebase; It is one of the most helpful features of Git. It fetches the latest commits from the master branch and puts our code on top of that. Thus, it maintains a clean history of the project.

Benefits of Git

A version control application allows us to **keep track** of all the changes that we make in the files of our project. Every time we make changes in files of an existing project, we can push those changes to a repository. Other developers are allowed to pull your changes from the repository and continue to work with the updates that you added to the project files.

Some **significant benefits** of using Git are as follows:



- **Saves Time**

  Git is lightning fast technology. Each command takes only a few seconds to execute so we can save a lot of time as compared to login to a GitHub account and find out its features.

- **Offline Working**

  One of the most important benefits of Git is that it supports **offline working**. If we are facing internet connectivity issues, it will not affect our work. In Git, we can do almost everything locally. Comparatively, other CVS like SVN is limited and prefer the connection with the central repository.

- **Undo Mistakes**

  One additional benefit of Git is we can **Undo** mistakes. Sometimes the undo can be a savior option for us. Git provides the undo option for almost everything.

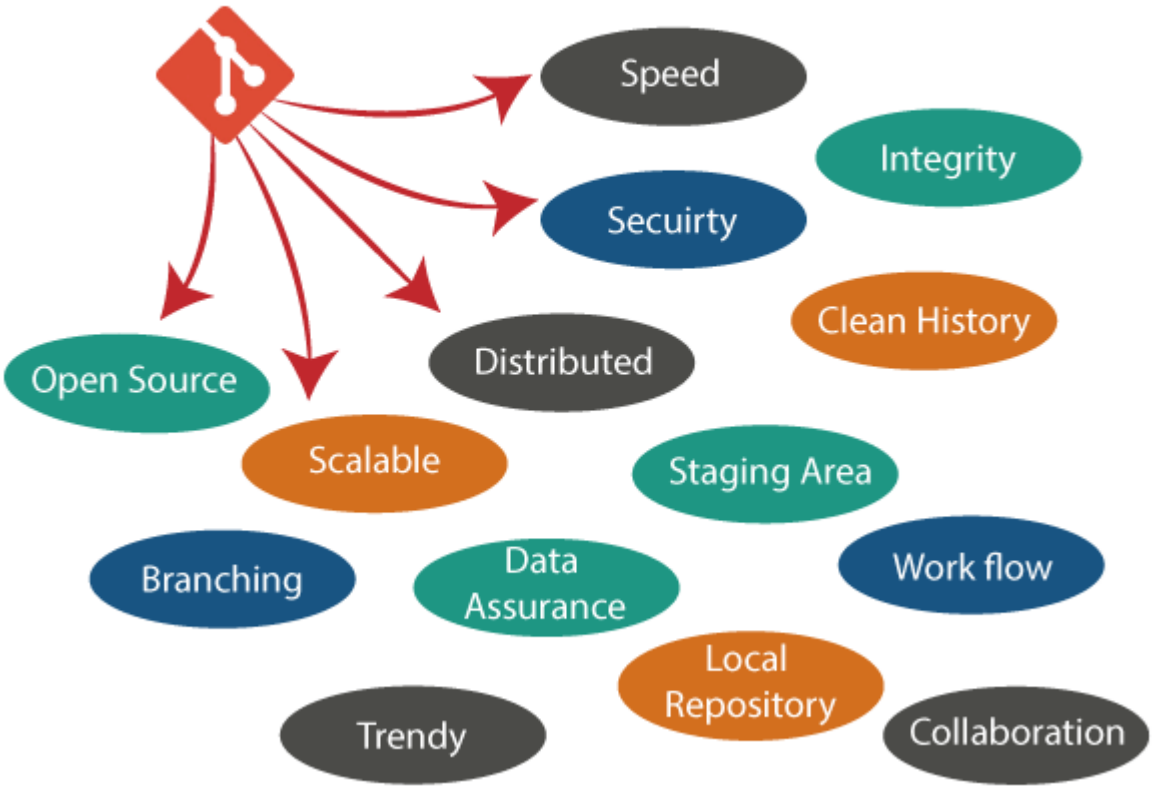
- **Track the Changes**

  Git facilitates with some exciting features such as **Diff, Log,** and **Status**, which allows us to track changes so we can **check the status, compare** our files or branches.

Why Git?

We have discussed many **features** and **benefits** of Git that demonstrate the undoubtedly Git as the **leading version control system**. Now, we will discuss some other points about why should we choose Git.

Why Git?


Speed
Integrity
Secuirty
Clean History
Distributed
Open Source
Scalable
Staging Area
Branching
Data Assurance
Work flow
Local Repository
Trendy
Collaboration

- **Git Integrity**

  Git is **developed to ensure** the **security** and **integrity** of content being version controlled. It uses checksum during transit or tampering with the file system to confirm that information is not lost. Internally it creates a checksum value from the contents of the file and then verifies it when transmitting or storing data.

- **Trendy Version Control System**

  Git is the **most widely used version control system**. It has **maximum projects** among all the version control systems. Due to its **amazing workflow** and features, it is a preferred choice of developers.

- **Everything is Local**

  Almost All operations of Git can be performed locally; this is a significant reason for the use of Git. We will not have to ensure internet connectivity.

- **Collaborate to Public Projects**

  There are many public projects available on the GitHub. We can collaborate on those projects and show our creativity to the world. Many developers are collaborating on public projects. The collaboration allows us to stand with experienced developers and learn a lot from them; thus, it takes our programming skills to the next level.

- **Impress Recruiters**

  We can impress recruiters by mentioning the Git and GitHub on our resume. Send your GitHub profile link to the HR of the organization you want to join. Show your skills and influence them through your work. It increases the chances of getting hired.

Prerequisites

Git is not a programming language, so you should have the basic understanding of Windows commands only.

What is GitHub?

GitHub is a Git repository hosting service. GitHub also facilitates with many of its features, such as access control and collaboration. It provides a Web-based graphical interface.

GitHub is an American company. It hosts source code of your project in the form of different programming languages and keeps track of the various changes made by programmers.

It offers both **distributed version control and source code management (SCM)** functionality of Git. It also facilitates with some collaboration features such as bug tracking, feature requests, task management for every project.

Features of GitHub

GitHub is a place where programmers and designers work together. They collaborate, contribute, and fix bugs together. It hosts plenty of open source projects and codes of various programming languages.

Some of its significant features are as follows.

- Collaboration
- Integrated issue and bug tracking
- Graphical representation of branches
- Git repositories hosting
- Project management
- Team management
- Code hosting

- Track and assign tasks
- Conversations
- Wikisc

Benefits of GitHub

GitHub can be separated as the Git and the Hub. GitHub service includes access controls as well as collaboration features like task management, repository hosting, and team management.

The key benefits of GitHub are as follows.

- It is easy to contribute to open source projects via GitHub.
- It helps to create an excellent document.
- You can attract recruiter by showing off your work. If you have a profile on GitHub, you will have a higher chance of being recruited.
- It allows your work to get out there in front of the public.
- You can track changes in your code across versions.

Git vs GitHub

Git is an open-source distributed version control system which is available for everyone at zero cost. It is designed to handle minor to major projects with speed and efficiency. It is developed to co-ordinate the work

among programmers. The version control allows you to track and work together with your team member at the same workspace.

While **GitHub is a Git repository hosting service**. It is a web-based service. GitHub facilitates with all of the features of distributed version control and source code management (SCM) functionality of Git. It also supports some of its characteristics in a single software tool.

To better understand the similarities and differences between Git and GitHub, look at the following points.

Git	GitHub
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 vs SVN

Apache Subversion or **SVN is one of the most popular centralized version control systems**. Now, SVN's popularity is on the decrease, but there are still millions of projects stored in it. It can continue to be actively maintained by an open-source community. In SVN, you can check out a single version of the repository. It stores data in a central server. The drawback of the SVN is, it has the entire history on a local repository which

limits you. You can only do commits, diffs, logs, branches, merges, file annotations, etc.

While, **Git is a popular distributed version control system**, which means that you can clone your repository. Thus you can get a complete copy of your entire history of that project. This means you can access all your commits.

Git has more advantages than SVN. It is much better for those developers who are not always connected to the master repository. Also, it is much faster than SVN.

To better understand the differences between Git and Subversion. Let's have a look at following significance points.

Git	SVN
It's a distributed version control system.	It's a Centralized version control system
Git is an SCM (source code management).	SVN is revision control.
Git has a cloned repository.	SVN does not have a cloned repository.
The Git branches are familiar to work. The Git system helps in merging the files quickly and also assist in finding the unmerged ones.	The SVN branches are a folder which exists in the repository. Some special commands are required For merging the branches.
Git does not have a Global revision number.	SVN has a Global revision number.
Git has cryptographically hashed contents that protect the contents from repository corruption taking	SVN does not have any cryptographically hashed contents.

place due to network issues or disk failures.	
Git stored content as metadata.	SVN stores content as files.
Git has more content protection than SVN.	SVN's content is less secure than Git.
Linus Torvalds developed git for Linux kernel.	CollabNet, Inc developed SVN.
Git is distributed under GNU (General public license).	SVN is distributed under the open-source license.

Git Version Control System

A version control system is a software that tracks changes to a file or set of files over time so that you can recall specific versions later. It also allows you to work together with other programmers.

The version control system is a collection of software tools that help a team to manage changes in a source code. It uses a special kind of database to keep track of every modification to the code.

Developers can compare earlier versions of the code with an older version to fix the mistakes.

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.

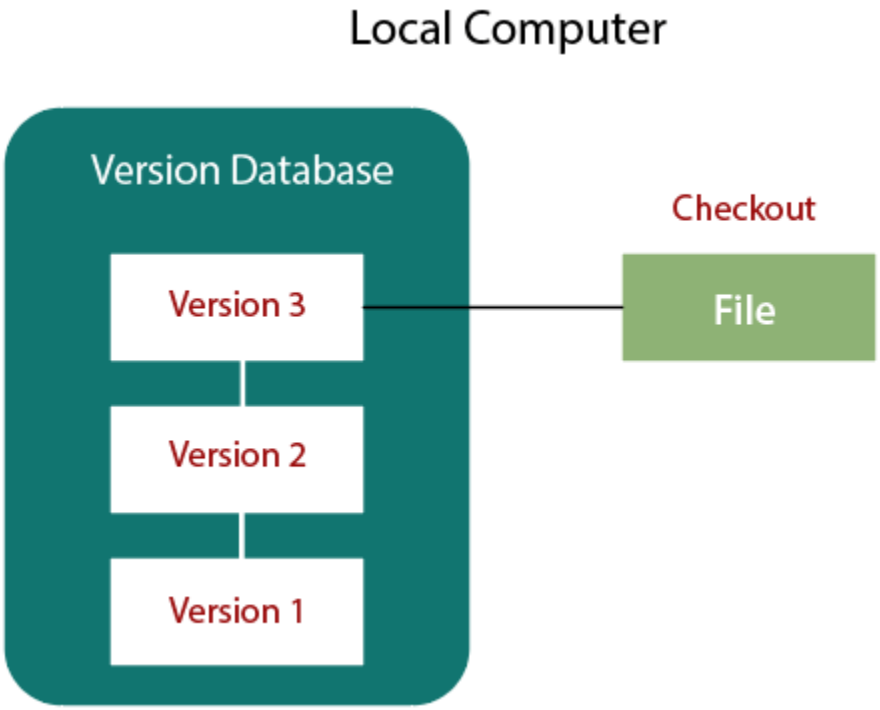
Some key benefits of having a version control system are as follows.

- Complete change history of the file
- Simultaneously working
- Branching and merging
- Traceability

Types of Version Control System

- Localized version Control System
- Centralized version control systems
- Distributed version control systems

Localized Version Control Systems



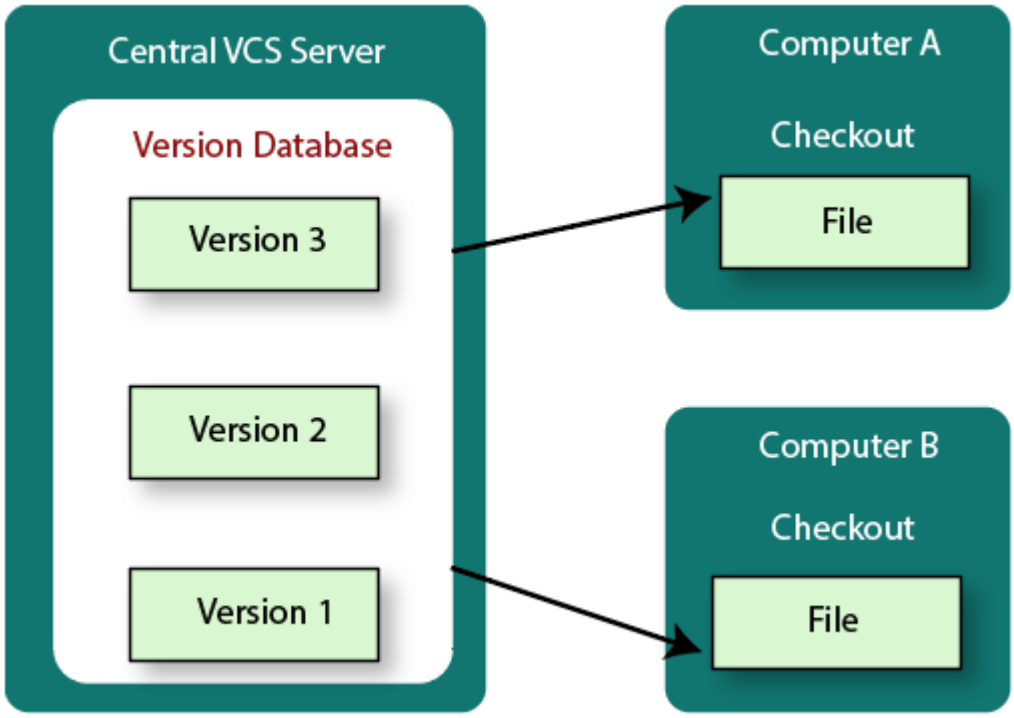
The localized version control method is a common approach because of its simplicity. But this approach leads to a higher chance of error. In this approach, you may forget which directory you're in and accidentally write to the wrong file or copy over files you don't want to.

To deal with this issue, programmers developed local VCSs that had a simple database. Such databases kept all the changes to files under revision control. A local version control system keeps local copies of the files.

The major drawback of Local VCS is that it has a single point of failure.

Centralized Version Control System

The developers needed to collaborate with other developers on other systems. The localized version control system failed in this case. To deal with this problem, Centralized Version Control Systems were developed.



These systems have a single server that contains the versioned files, and some clients to check out files from a central place.

Centralized version control systems have many benefits, especially over local VCSs.

- Everyone on the system has information about the work what others are doing on the project.

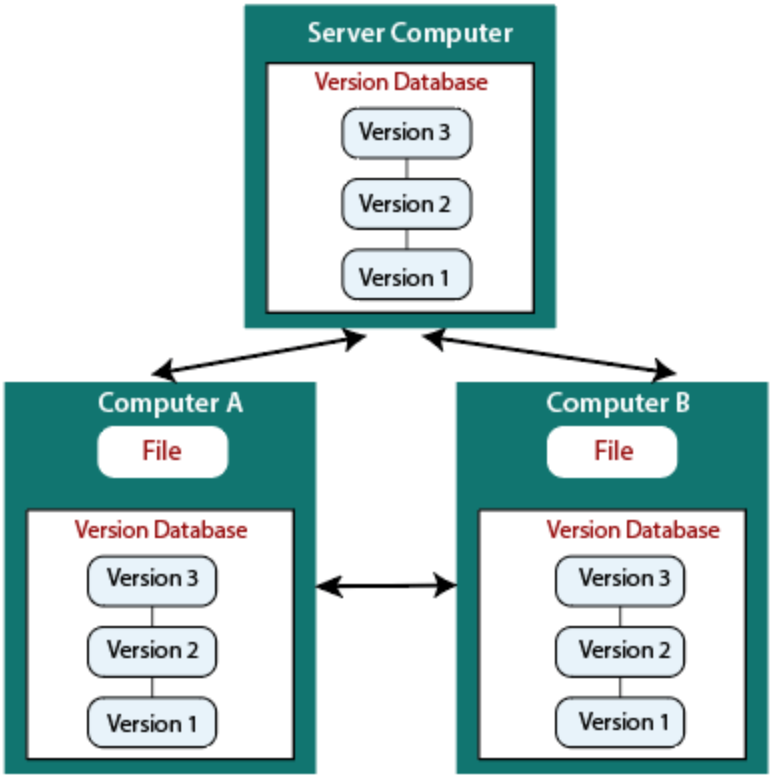
- Administrators have control over other developers.
- It is easier to deal with a centralized version control system than a localized version control system.
- A local version control system facilitates with a server software component which stores and manages the different versions of the files.

It also has the same drawback as in local version control system that it also has a single point of failure.

Distributed Version Control System

Centralized Version Control System uses a central server to store all the database and team collaboration. But due to single point failure, which means the failure of the central server, developers do not prefer it. Next, the Distributed Version Control System is developed.

In a Distributed Version Control System (such as Git, Mercurial, Bazaar or Darcs), the user has a local copy of a repository. So, the clients don't just check out the latest snapshot of the files even they can fully mirror the repository. The local repository contains all the files and metadata present in the main repository.



DVCS allows automatic management branching and merging. It speeds up of most operations except pushing and pulling. DVCS enhances the ability to work offline and does not rely on a single location for backups. If any server stops and other systems were collaborating via it, then any of the client repositories could be restored by that server. Every checkout is a full backup of all the data.

These systems do not necessarily depend on a central server to store all the versions of a project file.

Difference between Centralized Version Control System and Distributed Version Control System

Centralized Version Control Systems are systems that use **client/server** architecture. In a centralized Version Control System, one or more client systems are directly connected to a central server. Contrarily the Distributed Version Control Systems are systems that use **peer-to-peer** architecture.

There are many benefits and drawbacks of using both the version control systems. Let's have a look at some significant differences between Centralized and Distributed version control system.

Centralized Version Control System	Distributed Version Control System
In CVCS, The repository is placed at one place and delivers information to many clients.	In DVCS, Every user has a local copy of the repository in place of the central repository on the server-side.
It is based on the client-server approach.	It is based on the client-server approach.
It is the most straightforward	It is flexible and has emerged with

system based on the concept of the central repository.	the concept that everyone has their repository.
In CVCS, the server provides the latest code to all the clients across the globe.	In DVCS, every user can check out the snapshot of the code, and they can fully mirror the central repository.
CVCS is easy to administrate and has additional control over users and access by its server from one place.	DVCS is fast comparing to CVCS as you don't have to interact with the central server for every command.
The popular tools of CVCS are **SVN** (Subversion) and **CVS**.	The popular tools of DVCS are **Git** and **Mercurial**.
CVCS is easy to understand for beginners.	DVCS has some complex process for beginners.
If the server fails, No system can access data from another system.	if any server fails and other systems were collaborating via it, that server can restore any of the client repositories

Git Cheat Sheet

1. Git configuration

- **Git config**

  Get and set configuration variables that control all facets of how Git looks and operates.

  Set the name:

  $ git config --global user.name "Shekhar"

  Set the email:

  $ git config --global user.email "ltionlinemumbai@gmail.com"

  Set the default editor:

  $ git config --global core.editor Vim

  Check the setting:

  $ git config -list

- **Git alias**

  Set up an alias for each command:

  $ git config --global alias.co checkout

  $ git config --global alias.br branch

$ git config --global alias.ci commit

$ git config --global alias.st status

2. Starting a project

- **Git init**

  Create a local repository:

  # $ git init

- **Git clone**

  Make a local copy of the server repository.

  $ git clone

3. Local changes

- **Git add**

  Add a file to staging (Index) area:

  ## $ git add Filename

  Add all files of a repo to staging (Index) area:

  $ git add*

- # Git commit

  Record or snapshots the file permanently in the version history **with a message**.

  $ git commit -m " Commit Message"

4. Track changes

- **Git diff**

  Track the changes that have not been staged: $ git diff

  Track the changes that have staged but not committed:

  $ git diff --staged

  Track the changes after committing a file:

  $ git diff HEAD

  Track the changes between two commits:

  $ git diff Git Diff Branches:

  $ git diff < branch 2>

- **Git status**

  Display the state of the working directory and the staging area.

  $ git status

- **Git show Shows objects:**

  $ git show

5. Commit History

- **Git log**

  Display the most recent commits and the status of the head:

  $ git log

  Display the output as one commit per line:

  $ git log -oneline

  Displays the files that have been modified:

$ git log -stat

Display the modified files with location:

$ git log -p

- **Git blame**

  Display the modification on each line of a file:

  $ git blame <file name>

6. Ignoring files

- **.gitignore**

  Specify intentionally untracked files that Git should ignore. Create

  .gitignore:

  $ touch .gitignore List the ignored files:

  $ git ls-files -i --exclude-standard

7. Branching

- **Git branch Create branch:**

  ### $ git branch List Branch:

  $ git branch --list Delete a Branch:

  $ git branch -d Delete a remote Branch:

  $ git push origin -delete Rename Branch:

  ### $ git branch -m

- **Git checkout**

  Switch between branches in a repository.

Switch to a particular branch:

$ git checkout

Create a new branch and switch to it:

$ git checkout -b Checkout a Remote branch:

$ git checkout

- **Git stash**

  Switch branches without committing the current branch. Stash current work:

  $ git stash

  Saving stashes with a message:

  $ git stash save ""

  Check the stored stashes:

  $ git stash list

  Re-apply the changes that you just stashed:

  $ git stash apply

  Track the stashes and their changes:

  $ git stash show

  Re-apply the previous commits:

  $ git stash pop

  Delete a most recent stash from the queue:

  $ git stash drop

  Delete all the available stashes at once:

  $ git stash clear

Stash work on a separate branch:

$ git stash branch

- **Git cherry pic**

  Apply the changes introduced by some existing commit:

  $ git cherry-pick

8. Merging

- **Git merge**

  Merge the branches:

  ### $ git merge

  Merge the specified commit to currently active branch:

  $ git merge

- **Git rebase**

  Apply a sequence of commits from distinct branches into a final commit.

  $ git rebase

  Continue the rebasing process:

  $ git rebase -continue Abort the rebasing process:

  $ git rebase --skip

- **Git interactive rebase**

  Allow various operations like edit, rewrite, reorder, and more on existing commits.

  $ git rebase -i

9. Remote

- **Git remote**

  Check the configuration of the remote server:

  $ git remote -v

  Add a remote for the repository:

  $ git remote add Fetch the data from the remote server:

  $ git fetch

  Remove a remote connection from the repository:

  $ git remote rm

  Rename remote server:

  $ git remote rename

  Show additional information about a particular remote:

  $ git remote show

  Change remote:

  $ git remote set-url

- **Git origin master**

  Push data to the remote server:

  $ git push origin master Pull data from remote server:

  $ git pull origin master

10. Pushing Updates

- **Git push**

  Transfer the commits from your local repository to a remote server.

Push data to the remote server:

$ git push origin master Force push data:

$ git push -f

Delete a remote branch by push command:

$ git push origin -delete edited

11. Pulling updates

- **Git pull**

  Pull the data from the server:

  $ git pull origin master

  Pull a remote branch:

  $ git pull

- **Git fetch**

  Download branches and tags from one or more repositories. Fetch the remote repository:

  $ git fetch< repository Url> Fetch a specific branch:

  $ git fetch

  Fetch all the branches simultaneously:

  $ git fetch -all

  Synchronize the local repository:

  $ git fetch origin

12. Undo changes

- **Git revert**

  Undo the changes:

  $ git revert

  Revert a particular commit:

  $ git revert

- **Git reset**

  Reset the changes:

  $ git reset -hard

  $ git reset -soft:

  $ git reset --mixed

13. Removing files

- **Git rm**

  Remove the files from the working tree and from the index:

  $ git rm <file Name>

  Remove files from the Git But keep the files in your local repository:

  $ git rm --cached