Lesson:

Version Control System

PW SKILLS

Topics

- Introduction to Version Control System
- Need of Version Control System
- Working of a Version Control System
- Introduction to git
- How to install git?
- Why use git?
- Alternatives of git

Introduction to version control system

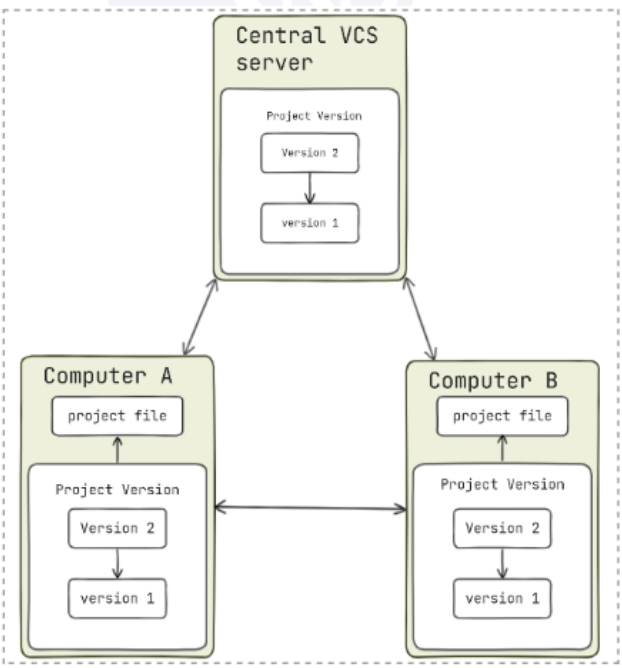
Version control system (VCS), also known as Source Code Management (SCM) is a fundamental tool used in software development and other collaborative projects to manage and track changes to files over time. VCSs are an integral part of the development process and a must-have tool for both individual developers and teams as well.

Version control is a system that records changes to a file or set of files over time. It allows you to track the history of changes, collaborate with others, and manage multiple versions of a project. Version control is commonly used in software development, but it can also be applied to any context where files need to be tracked and shared.

There are different types of version control systems, Some of the widely and commonly used VCSs can be of -

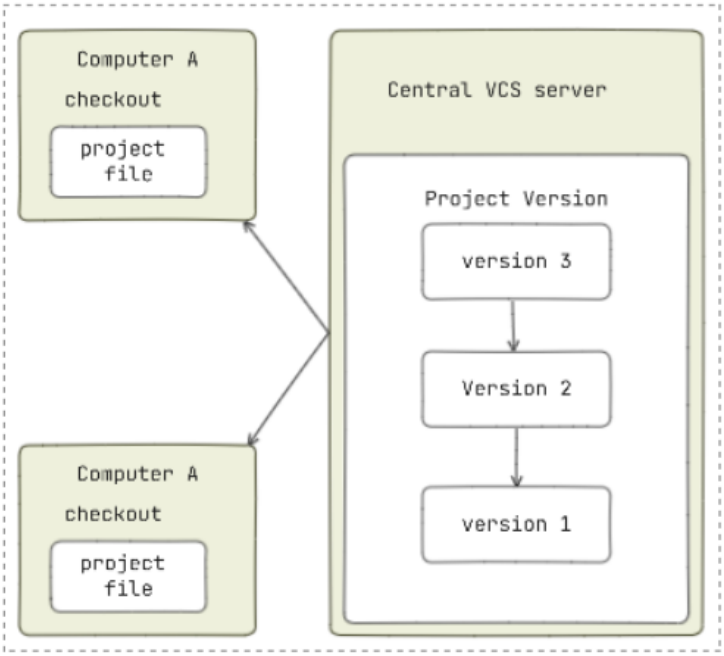
1. Distributed Version Control System (DVCS) - A Distributed Version Control System (DVCS) is a type of version control system that allows multiple copies of a repository to exist, each of which contains the full history of the project.

Overview flow diagram of Distributed version control system



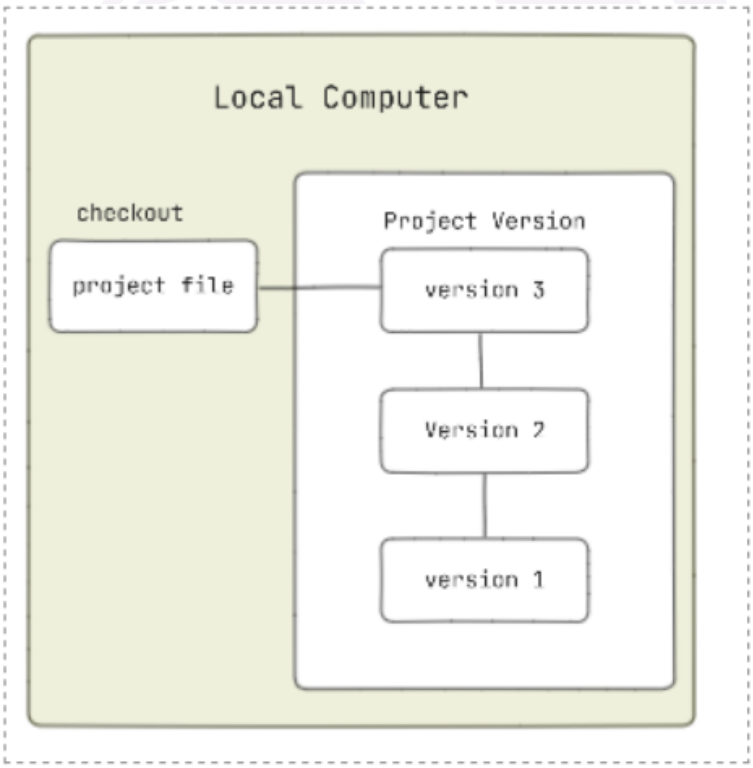
2. Centralized Version Control System (CVCS) - A Centralized Version Control System (CVCS) is a type of version control system where there is a central server that stores the main repository containing the entire history of the project.

Overview flow diagram of Central Version Control System



3. Local Version Control System - A Local Version Control System is the simplest form of version control and is designed for individual use or small projects

Overview flow diagram of Local version control system



Need of Version Control System

Version Control Systems (VCS) are crucial tools in software development, and understanding their importance is essential, especially for beginners.

Here's an explanation of the need for a version control system:

- Tracking History changes - A VCS records all changes to project files, allowing developers to track why and when changes occurred, aiding debugging and project understanding.
- Collaboration or Parallel development - In software development, team members often work simultaneously on the same project. VCS enables collaboration through branches, simplifying the combination of changes.
- Backup and Disaster Recovery - VCS serves as a backup system for project code. Even if a developer's local copy is lost or damaged, the project's history on the central server ensures data safety.
- Code Reviews and Quality Assurance - VCS enables code reviews, allowing team members to assess code quality, adherence to standards, and issues before merging it.
- Experimentation and Rollback- Developers can experiment in separate branches without impacting the main project. If an experiment fails, branches can be easily discarded, returning to the previous state.
- Release Management - VCS aids software release management by tagging specific code versions, ensuring consistent and reliable software releases.
- Traceability and Documentation - VCS offers an audit trail, documenting changes and providing context for project history and decisions.
- Conflict Resolution - When developers simultaneously modify the same code, VCS tools help resolve conflicts systematically, highlighting necessary changes.
- Open Source and Collaboration Beyond Teams - VCS tools are vital for open-source projects, enabling global collaboration and maintaining code quality.

Working of a Version Control System

Understanding how a version control system works is essential for managing and tracking changes in software development.

Here is a simple explanation of how a VCS operates

- Repository - A VCS starts with a central storage location called a "repository." This repository holds all the project files and their entire history.
- Local Copy - Each developer working on the project creates a personal "local copy" of the repository on their computer. This local copy contains all the project files and their history.

- Making Changes - Developers can make changes to the project files in their local copy. These changes can include adding, modifying, or deleting files and code.

- Committing changes - After making changes, developers create a "commit." A commit is like taking a snapshot of the project at that moment. It records what changes were made and includes a message explaining the purpose of the changes.

- Tracking Changes - The VCS continuously tracks the differences between commits, creating a history of changes over time. This history helps developers understand how the project evolved and why specific changes were made.

- Branching and Merging - Developers can create separate "branches" to work on specific features or fixes independently. Once a branch is complete, its changes can be "merged" back into the main project, combining the changes from different branches.

- Conflict Resolution - When multiple developers make conflicting changes (e.g., editing the same line of code), the VCS helps identify these conflicts, allowing developers to resolve them systematically

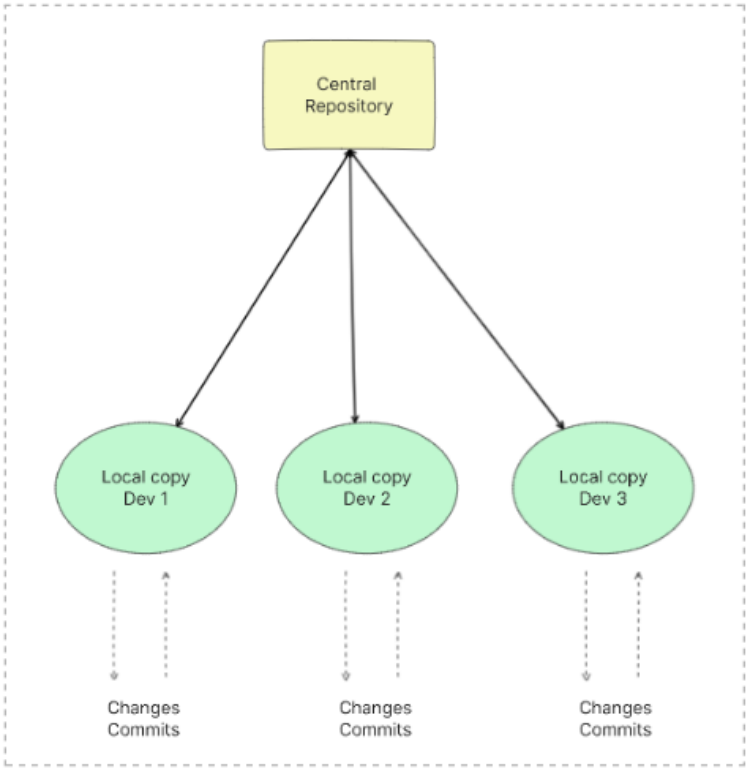
- Collaboration - Developers can share their commits with others by "pushing" their changes to the central repository. They can also "pull" changes made by others to update their local copies, enabling collaboration on a shared codebase.

- History and versioning - The VCS maintains a detailed history of all commits and changes made to the project. Developers can use this history to understand the context behind each modification and to roll back to a previous state if needed.

- Tagging and Releases - Developers can "tag" specific commits to mark significant milestones or releases. These tags help identify stable versions of the project.

- Remote Repository - In many cases, VCS systems support "remote repositories" hosted on servers. These remote repositories allow for distributed collaboration, enabling developers to share their work with others even if they are not in the same physical location.

Overview Flow diagram of Working of Version control system

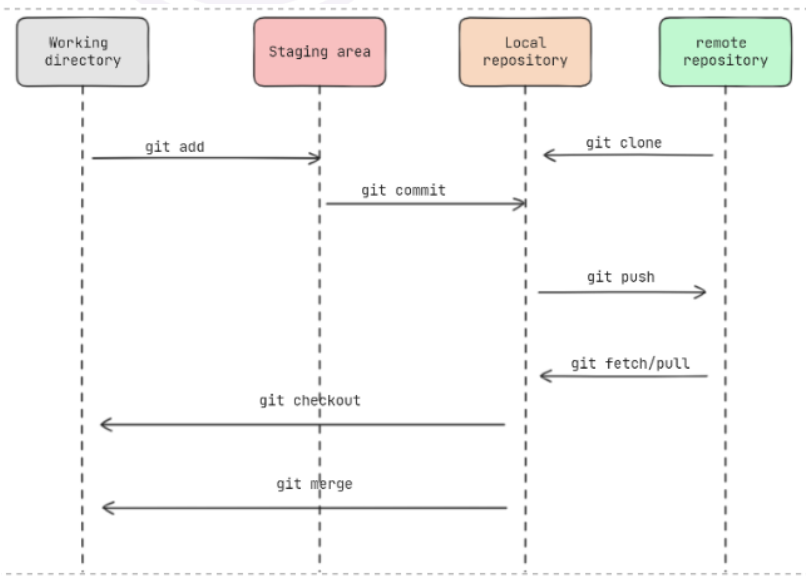


Introduction to Git

Git is a widely used distributed version control system (DVCS) that is essential in modern software development. It was created by Linus Torvalds in 2005 and has since become the industry standard for source code management. Git is known for its speed, flexibility, and powerful branching capabilities.

Git is a VCS, which means it helps developers manage changes to their code over time. It allows you to track modifications, collaborate with others, and maintain a history of the project.

The basic flow of Git



Note - More details about the keyword used in the diagram will be discussed in the next document

Git has three main states that your files can reside in committed, modified, and stage. Committed means that the data is safely stored in your local storage. Modified means that you have changed the file but have not committed it to your remote storage yet. Staged means that you have marked a modified file in its current version to go into your next commit snapshot.

How to install Git?

For the installation of Git on any system, certain specific instructions should be followed depending on a particular operating system-

Windows -

- Visit the official Git for Windows website i.e**. Link**
- Download the latest installer (64-bits is recommended)
- Run the installer and follow the installation wizard
- During the installation, one can choose various configuration options. The default settings are usually fine for most users.

Mac -
For installing it Mac OS Homebrew should be installed from the official Homebrew site i.e**. Link**
After installing the Homebrew run the following command below to install the git.

```
Unset
brew install git
```

After successful installation, the first thing you should do is to set your username and email address. This is important because every Git commit uses this information, and it's immutably backed into the commits you pass around.

Run the below command provides your name and email address to set your name and email address.

```
Unset
 git config --global user.name "John Doe"
git config --global user.email "johndoe@example.com"
```

To check the git setting run the command below -

```
Unset
git config --list
```

In case, if help is needed while using Git, there are three ways to get the manual page (manpage) help for any of the git commands.
example of using help command

```
Unset
# syntax
git help <verb>
git help config # example to get help on the "config" command

git <verb> --help
git help add # example to get help on the "add" command
```

Why use Git?

Git is used because it is highly regarded for its role as a distributed version control system, as it not only effectively fulfills all the essential features and requirements expected of a version control system but also offers a robust and versatile framework that empowers developers to collaboratively manage source code, track changes, and seamlessly coordinate their work across diverse teams and locations while providing robust branching, merging, and historical tracking capabilities.

Alternatives of Git

Git is one of the most widely used Version Control systems used by software developers. However, there are several alternative version control systems available that offer similar features and functionality to Git.

They are -

- Mercurial - Mercurial is a free, open-source version control system that is similar to Git in many ways. It is known for its simplicity and speed and is popular among developers who prefer a lightweight tool.

- Subversion - Subversion (also known as SVN) is another popular version control system that is similar to Git. It is known for its reliability and simplicity and is often used in large organizations.

- Concurrent Version System - CVS (Concurrent Versions System) is an older version control system that is still used by some developers today. It is known for its simplicity and ease of use but lacks some of the more advanced features found in newer systems like Git and Mercurial.