Version Control System

A version control system (VCS), also known as a source control system, is a software tool that helps software developers manage changes to source code over time. It allows developers to track changes to the code, collaborate on development, and maintain a history of all changes made to the codebase.

Type of VCS (Version Control System)

CVCS (Centralized Version Control Systems)

DVCS

What is DVCS

A distributed version control system (DVCS) is a type of version control system where every developer has a copy of the entire codebase, including its full history.

In a DVCS, developers can make changes to their local copy of the code and then push those changes to a central repository when they are ready to share their work with others. Similarly, when other developers make changes to the central repository, each developer can pull those changes down to their local copy and merge them with their own changes.

One of the main advantages of DVCS is that it provides greater flexibility and redundancy. With a copy of the full repository on each developer's machine, there is less reliance on a central server, making it easier to work offline and reducing the risk of data loss. Additionally, DVCS makes it easier for multiple teams to work on different parts of the codebase in parallel, while still being able to merge their changes together.

Example - Git

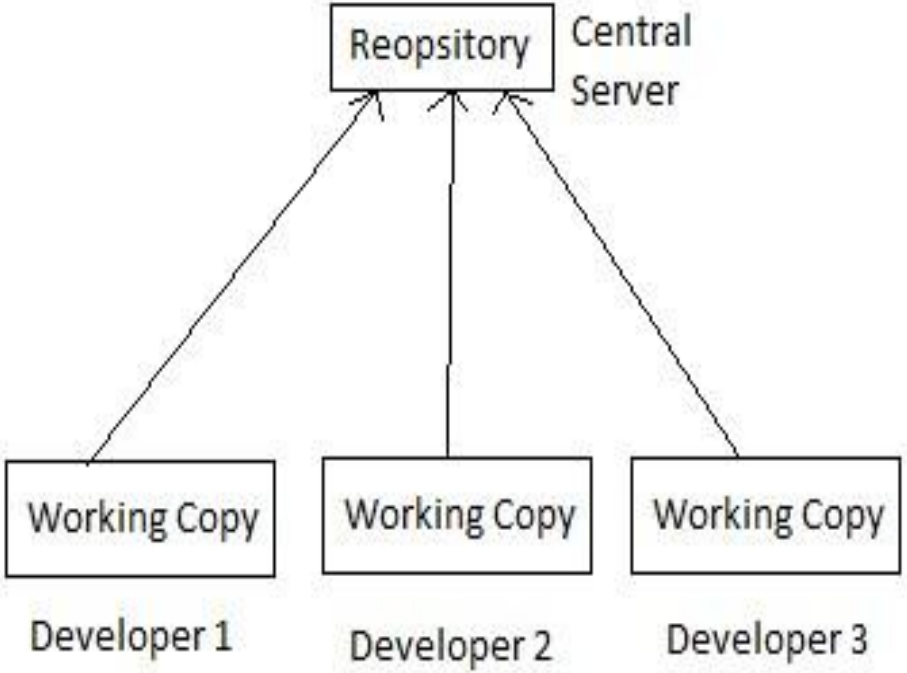
What is CVCS

A centralized version control system (CVCS) is a type of version control system in which there is a single central repository that stores the entire codebase and its history. Developers work on local copies of the code and use the CVCS to synchronize their changes with the central repository.

In a CVCS, developers typically check out a copy of the codebase from the central repository, work on it locally, and then commit their changes back to the central repository when they are ready to share their work with others. This ensures that everyone is working on the same codebase, with a single source of truth for the entire project.

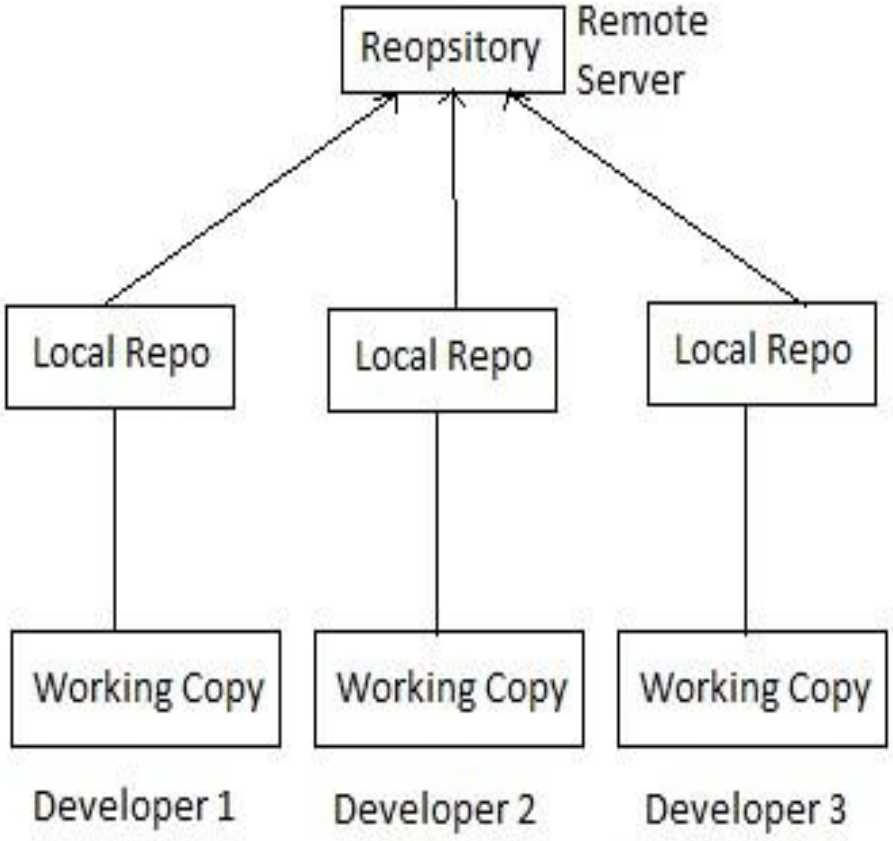
Examples of centralized version control systems include Subversion (SVN) and Perforce.

CVCS provides some benefits, such as simpler management and control of the codebase, and a centralized location for backups and disaster recovery. However, it also has some drawbacks. For example, it can be slower to work with, especially when working on large codebases or with geographically dispersed teams. Additionally, a single point of failure can be a risk, since if the central repository becomes unavailable, developers will be unable to access the codebase.



E.g. SVN

Centralized Version Control System



E.g. GIT

Distributed Version Control System

What is Git

Git is a distributed version control system that helps track changes made to a codebase over time. It was created by Linus Torvalds in 2005 for managing the development of the Linux kernel.

Git allows multiple developers to work on the same project simultaneously, with each developer having their own copy of the repository on their local machine. Changes made by each developer can be committed to their local repository and then merged with the main repository to create a new version of the project. This makes it easy to keep track of changes, revert to previous versions, and collaborate on code with others.

Git also includes powerful branching and merging functionality, which allows developers to work on separate branches of the codebase and then merge those branches back into the main project. This makes it easy to test new features or make experimental changes without affecting the main codebase.

Git is widely used in software development and has become the de facto standard for version control in the industry. It is a free and open-source tool that can be used on most operating systems, including Windows, macOS, and Linux.

Commands

git init: initializes a new Git repository in a directory

git add: adds files to the staging area (the area where changes are prepared to be committed)

git commit: saves changes to the local repository with a commit message

git clone: creates a copy of a remote repository on your local machine

git pull: updates your local repository with changes from a remote repository

git push: pushes changes from your local repository to a remote repository

git branch: lists or creates branches in the repository

git checkout: switches between different branches or commits in the repository

git merge: merges changes from one branch into another

git status: shows the status of the working directory and the staging area

git log: shows a history of commits in the repository

What is Github

GitHub is a web-based platform that provides hosting for Git repositories. It allows individuals and teams to easily collaborate on software development projects, and provides a range of features for managing code, tracking issues, and deploying software.

It provides a simple and intuitive web interface for managing Git repositories, making it easy to view changes, create new branches, and collaborate with others. GitHub also provides features for managing issues, tracking bugs, and creating pull requests, which are requests to merge changes made by a developer into the main codebase.

One of the most important features of GitHub is its social aspect. Developers can follow other developers and projects, and contribute to open-source projects hosted on the platform. This has helped to create a vibrant community of developers who share their knowledge and work together to create better software.

Branches

Branches in Git allow you to work on different versions of your project, test new things, and then combine them back into the main project when you're ready.

In Git, you merge branches to combine the changes made in one branch with another branch, typically bringing the changes from a feature branch into the main branch (often called the "master" or "main" branch). Merging is important for a few reasons:

Why merge

1. Incorporate New Features: When you work on a new feature or fix a bug in a feature branch, you want to bring those changes into the main branch so that your project benefits from the improvements. Merging allows you to add these new features or fixes to the main branch.

2. Keep the Project Up-to-Date: Projects evolve, and multiple people may be working on different branches simultaneously. Merging helps you keep the main branch up-to-date with the latest changes, ensuring that everyone's work is integrated smoothly.

3. Maintain a Clean History: Merging helps you maintain a clean and organized project history. Each merge represents a point where a set of changes was incorporated into the main branch, making it easier to track the progress of the project.

4. Collaboration: Git is often used for collaborative software development. Merging allows team members to share their work and collaborate on different aspects of a project while keeping everything in sync.

Pull Request

Pull request is a way for developers to propose changes, get feedback, and eventually merge their work into the main project or another branch in a controlled and collaborative manner. It's a crucial tool for maintaining code quality and facilitating teamwork in software development.