Version Control:

26/09/25 Satvik Jayaswal

1. *What is Version Control?:*

•As the name suggests, version control refers to the category of software tools that make it possible for the software team to look after the changes made to the source code. It is also known as Source Code Management(SCM) in broader terms.

•This system records all the changes made to a file so that a specific version may be rolled back if needed in the future.

•It is the role of the version control system to keep all the team members on the same page. It ensures that everyone on the team is working on the latest version of the files and, more importantly, that all of these people can work simultaneously on the same project.

•In simple words, it is essentially a record that keeps a track of every and any changes made to a document/ source code/other assests(mainly source code within the context of a software project for version control systems, SCM's can be regarded for a broader practice of managing) throughtout the peoject in question's lifetime. Allowing review of specific versions of the project if and when needed.

2 . *Why is Version Control important?:*

It ensures that changes to the code are tracked, allowing developers to revert to previous versions if needed and enabling simultaneous development by multiple team members.Version Control is crucial for both dvelopers and large teams for several key reasons:

*i.*History Tracking:

All changes, including who made them and when, are documented. If a mistake is introduced, the team can quickly identify the specific modification that caused the problem and return to a stable working version.

*ii.*Collaboration:

It enables numerous developers to work on the same project files simultaneously without stepping on each other's toes. The system includes tools for merging modifications and resolving disagreements.

*iii.*Safety and Backups:

Project history is stored in a central system. If a local machine fails or code is mistakenly deleted, the project can be completely restored.

*iv.*Branching and Experimenting:

Developers can build isolated environments (branches) to develop new features or address bugs without affecting the main source. Once the work is finished and

tested, the modifications can be safely merged back.

v. Auditing:

Provides a detailed audit trail of every development activities, crucial for regulated sectors and understanding code evolution.

3 . State and explain different types of Version Control:

Version control systems are generally categorized into three main types based on how they manage the history:

i. Local Version Control Systems (LVCS):

An LVCS stores the version history directly on the developer's local PC. This is the simplest type, and it uses a local disc database to track file modifications.

ii. Centralized Version Control Systems (CVCS):

CVCS stores all versioned files and history on a single, central server. Developers select files from this central archive, work on them, and then submit their modifications to the server.

iii. Distributed Version Control Systems (DVCS):

In a DVCS, each developer's machine has a complete copy of the archive, including its history. When a developer copies the archive, they receive not only the most recent files, but also the whole history log.

4 . Mention the important git commands and explain them:

A Git command instructs the Git version control system to do a specified action related to managing and tracking changes in a codebase. These commands are often used via a command-line interface (CLI) or terminal, allowing developers to communicate with Git repositories and manage various elements of version control.

- **git init**: Creates a new local Git repository in the current directory.
- **git clone [url]**: Creates a full copy of a remote repository on your local machine.
- **git status**: Displays the state of the working directory and staging area, showing modified, staged, and untracked files.
- **git add .**: Stages all changes in the current directory for the next commit.
- **git commit -m "message"**: Records the staged changes permanently into the repository history with a descriptive message.
- **git log**: Shows the chronological history of commits for the current branch.
- **git branch [name]**: Creates a new isolated line of development named **[name]**.
- **git checkout [branch]**: Switches the working

directory to the specified **[branch]** or commit.

- **git pull**: Fetches changes from the remote repository and automatically merges them into the current branch.
- **git push**: Uploads all local branch commits to the corresponding remote repository.
- **git'D**: Deletes a branch
- git -rm "-remote": Remove a remote depository
- Git fetch: fetches only changes from remote repository