Version Control: Git vs SVN (Subversion)

1. What is Version Control? Why do we use it?

Imagine a team of 4 developers (Dev A, B, C, D) working on a banking web application. Each dev is responsible for different features like login, money transfer, dashboard, and notifications.

Without version control:
- Each dev writes code on their local machine.
- They send files over email or shared drives.
- Merging changes becomes manual and messy.

With version control:
- All developers push their code to a central repository (e.g., GitHub).
- They can track changes, rollback, merge, and collaborate efficiently.

2. Without Version Control

Problem:
- All 4 developers work on the same project, but on different laptops.
- They email files to each other to update.
- Every dev has to manually merge others’ code.
- Leads to conflicts, duplicate effort, and wasted time.

Example:
- Dev A sends login feature update to Dev B.
- Dev B modifies it and sends it to Dev C.
- Dev C adds his changes and sends to Dev D.
- Dev D must now reconcile all code from A, B, and C — which is chaotic.

Result:
- 80% time is wasted managing code manually, and only 20% is spent on real development.

3. With Version Control (Git + Remote Repository)

Each developer:
- Clones the remote repo to their machine.
- Works on their feature in a branch.
- Pushes changes back to the remote repo.
- Can pull others' changes and merge efficiently.

Remote Repo (e.g., GitHub/GitLab) acts as the central source of truth.

Result:
- 80% time is used efficiently for development, only 20% on merging if needed.

4. Why Version Control?

- Track who changed what and when.
- Rollback to previous versions if needed.
- Collaborate safely without overwriting others' work.
- Branching allows isolated feature development.
- Auto-merging when possible.

5. Types of Version Control

A. Centralized Version Control (CVCS) - e.g., SVN
- Server stores the main codebase.
- Developers checkout only parts of the code.
- If the server is down, no collaboration.

B. Distributed Version Control (DVCS) - e.g., Git
- Every developer has the full copy of the repo.
- Can work offline, commit locally, and push later.
- Much safer and efficient for modern projects.

6. Git Architecture

Multiple Git clients (developers) push to a Git Remote Repo.
Everyone can pull latest code and push their changes.
Code is distributed and safe.
Collaboration becomes seamless.

7. Summary

Without Git:
- Manual merging
- File sharing via email
- High risk of conflicts
- Poor tracking
- Wasted time

With Git:
- Automated merges
- Push/pull from central repo

- Branching handles parallel work
- Full history & authorship
- Productive collaboration

8. Git vs SVN (Subversion)

Feature	Git	SVN (Subversion)
Type	Distributed Version Control System (DVCS)	Centralized Version Control System (CVCS)
Repository Location	Every developer has a full copy of the repository (local + remote)	Only the central server has the full repository; developers work with local working copies
Offline Work	Fully functional offline (commit, view history, create branches)	Limited offline capabilities; most operations need server access
Speed	Faster (most operations are local)	Slower (depends on server communication)
Branching	Lightweight, fast, and heavily used	Heavier, slower, and less commonly used
Merging	Powerful and frequent merging tools	Merging is harder and more manual
Performance	Excellent with large projects	Slower with very large repositories
Security	Repository is fully copied to each developer, so data loss is unlikely	Single point of failure (if central repo is lost, history is gone)
Example Use	Open source projects (e.g., Linux Kernel, GitHub projects)	Older enterprise projects, legacy systems
Tooling	Git CLI, GitHub, GitLab, Bitbucket, etc.	