TOPIC : GIT INSTALLATION

NAME : MULAGADA DILEEP KUMAR

EID : PCIS462

Types of Version Control Systems (VCS)

1.1 Centralized Version Control System (CVCS):-

CVCS is a version control system where a central server stores all versions of the code. Developers commit their changes to this central repository.

Advantages of CVCS:

- **Single Source of Truth:** All files are stored in one central repository.
- **Easier Administration:** Simplified user and access management.
- **Less Storage Required:** Only one central repository is maintained.

Disadvantages of CVCS:

- **Single Point of Failure:** If the central server crashes, all data is lost.
- **Network Dependency:** Requires an active connection to commit changes.
- **Limited Branching:** Not as flexible as DVCS.

1.2 Distributed Version Control System (DVCS):-

DVCS allows every developer to have a complete copy of the repository, including the full history of changes.

Advantages of DVCS:

- **No Single Point of Failure:** Every developer has a full copy of the repository.
- **Offline Work:** Developers can commit changes without an internet connection.
- **Efficient Branching and Merging:** Easier to create and manage branches.

Disadvantages of DVCS:

- **More Storage Required:** Each developer has a full copy of the repository.
- **Steeper Learning Curve:** Can be complex for beginners.

---

2. Git: Introduction and Commands

Git is a popular DVCS used for tracking code changes.

1 How to Create a Repository

- **Initialize a new Git repository:** Use the git init command in a project directory to turn it into a Git repository.
- **Clone an existing repository:** Use git clone followed by the repository URL to download an existing repository from a remote source.

2 Adding Files to Git

- **Add a single file:** Use git add filename to stage a specific file for commit.
- **Add all files in the directory:** Use git add . to stage all changes in the directory.

3 Committing Changes

- **Commit with a message:** Use git commit -m "Commit message" to save staged changes with a description.

4 Checking Status

- **Check the status of the working directory:** Use git status to see which files are staged, modified, or untracked.

5 Checking Commit History

- **Show commit history:** Use git log to view the commit history with details like author and timestamp.

6 Pushing Changes to Remote Repository

- **Push changes to the remote repository:** Use git push origin branch_name to upload local commits to a remote repository.

7 Pulling Changes from Remote Repository

- **Pull latest changes:** Use git pull origin branch_name to fetch and merge the latest changes from the remote repository.

8 Creating and Switching Branches

- **Create a new branch:** Use git branch branch_name to create a new branch.
- **Switch to a branch:** Use git checkout branch_name to move to another branch.
- **Create and switch to a branch simultaneously:** Use git checkout -b branch_name.

9 Merging Branches

- **Merge a branch into the current branch:** Use git merge branch_name to combine another branch into the current one.

10 Resolving Merge Conflicts

- **Edit conflicted files manually, then stage them using git add filename**.
- **Commit the resolved files using git commit -m "Resolved merge conflict"**.

---

3. Server-Client Model in Git

3.1 Server (Remote Repository)

A central server (e.g., GitHub, GitLab, Bitbucket) stores the repository.

- Example: GitHub hosts repositories that multiple developers can access.

3.2 Client (Local Repository)

Developers work on local repositories and sync changes with the remote server.

- Example: Developers clone a GitHub repository to their local machine and commit changes.

---

4. Real-Time Usage of Git in Development

4.1 Working in a Team

1. Clone the project repository.
2. Create a new feature branch.
3. Work on changes and commit regularly.
4. Push the branch to the remote repository.
5. Create a pull request and get a code review.
6. Merge changes into the main branch.

4.2 Handling Code Reviews and PRs

- Use **GitHub Pull Requests (PRs)** for collaboration.
- Request **reviews from peers** before merging.
- Maintain a **clean commit history**.

4.3 Continuous Integration and Deployment (CI/CD)

- **CI/CD pipelines** use Git repositories for automated testing and deployment.

- Example: GitHub Actions, Jenkins, or GitLab CI/CD.
