Module 5 : Version Control with Git & Project Implementation

Assignment 5.1 → Understanding Git Basics

1. Introduction to Version Control

(A) Definition and Significance of version Control Systems

A version control system (VCS) is a tool that helps manage changes to source code overtime. It allows multiple developers to collaborate on a project, track revisions, and revert to previous states if necessary. VCS ensures that changes are systematically recorded, providing a history of modifications and facilitating teamwork.

(B) Benefits of Utilizing Version Control for Software Development

(1) Collaboration : Multiple developers can work on the same project simultaneously without overwriting each other's work.

(2) History and Backup : Each change is recorded with a timestamp and a descriptive message, serving as a backup.

(3) Branching and Merging : Developers can create branches to work on new features or bug fixes without affecting the main codebase.

(4) Conflict Resolution : VCS help detect and resolve conflicts when different changes are made to the same part of the code.

(II) Core Concepts of Git

(A) Repositories: Local and Remote

- Local Repository: A version of the repository stored on the developer's machine.
- Remote Repository: A version of the repository hosted on a server, used for collaboration.

(B) Working Directory: Workspace for Project files

The working directory contains the files of your project that you are currently working on. These files can be in various states: untracked, modified, or staged.

(C) Staging Area (Index): Selecting Changes for Commits

The staging area or index, is a place where you can group changes before committing them. It allows you to prepare a snapshot of the project state.

(D) Commits: Capturing Project states with Descriptive Messages

A commit records changes to the repository. Each commit has a unique identifier and includes a descriptive message that explains the changes made.

(E) Branches: Divergent Development Paths within a Repository

Branches allow you to diverge from the main codebase to work on features, fixes, or experiments. The main branch is often called 'main' or 'master'.

(III.) Essential Git Commands

(A) Initialization : Creating a New Git Repository

Code →

```
git init
```

Creates a new Git repository in the current directory.

(B) Tracking Changes : Identifying Modified files

Code → git status

Shows the status of changes as untracked, modified or staged.

(C.) Staging and Committing : Preparing and Recording Changes

Code →

```
git add <file>
git commit -m "Descriptive message"
```

stages the specified file and commits the changes with a message.

(D.) Branching : Creating and Switching b/w Development Lines

Code →

```
git branch <branch-name>
git checkout <branch-name>
```

Creates a new branch and switches to it.

(E.) Merging : Integrating Changes from Different Branches

Code →

```
git checkout <target-branch>
git merge <source-branch>
```

Merges the changes from the source branch into the target branch.

(F.) Remote Repositories: Collaboration and Shared Workspaces

Code →

```
git remote add origin <repository-URL>
git push -u origin <branch-name>
git pull origin <branch-name>
```

Adds a remote repository, pushes changes, and pulls updates.

(IV.) Mastering Git Workflows

(A) Features Branch Workflow: Streamlined Development and Integration

(1.) Create a new branch for each feature or bug fix.

(2.) Work on the feature branch.

(3.) Merge the feature branch into the main branch when the work is complete.

(B.) Gitflow Workflow: Structured Approach for Large-Scale Projects

(1.) Main Branch: Contains production-ready code.

(2.) Develop Branch: Integrates features for the next release.

(3.) Feature Branches: For developing new features.

(4.) Release Branches: For preparing a new release.

(5.) Hotfix Branches: For emergency fixes on the production code.

(V.) Advanced Git Techniques

(A.) Resolving Merge Conflicts: Handling Conflicting changes

when conflicts arise during a merge:

code → git status

identify conflicting files and manually resolve them. Then:

code → git add <resolved-file>

git commit

Finalize the merge after resolving conflicts

(B.) Stashing Changes: Temporarily Shelving Uncommitted Work

code → git stash

git stash pop

Saves the current state and reverts to the previous commit.

'git stash pop' restores the stashed changes.

(C.) Using Tags: Annotating Specific Project Versions

code → git tag <tag-name>

git push origin <tag-name>

Creates a tag to mark a specific point in the repository's history, useful for releases.