Module 1: L2

Introduction to Git and Version Control

Git is a distributed version control system that allows developers to track changes, manage multiple versions, and collaborate on code projects. It's essential for organizing code changes, especially when working with teams or maintaining projects over time. Version control systems (VCS) like Git help manage code history, revert changes, create branches, and merge work from different contributors.

Key Concepts in Git

- **Repository**: A storage for your code, where all your project's files and the entire history of their changes are tracked.
- **Commit**: A snapshot of your changes. Commits are like saves that store the state of your project at a given time.
- **Branch**: A separate line of development. Git's branching system allows for multiple people to work on different features without affecting each other.
- **Merge**: Combines the changes from different branches.
- **Remote Repository**: An online version of your repository (e.g., on GitHub, GitLab) that others can access.
- **Push**: Uploads local changes to the remote repository.
- **Pull**: Fetches and integrates changes from the remote repository.

Basic Git Commands

1. Initialize a Git repository: **git init** This command initializes a new Git repository in your project folder.
2. Clone an existing repository: **git clone <repository-url>**
   Copies a remote repository to your local system.
3. Check status: **git status**
   Shows the current state of the repository, including modified files, untracked files, and staged changes.
4. Add changes: **git add <file>**
   Stages changes to a file. Use git add . to add all changes in the project.
5. Commit changes: **git commit -m "Commit message"**
   Saves your changes to the local repository with a message describing what was changed.
6. Push changes to remote: **git push origin <branch>**
   Pushes commits from your local branch to a remote branch.
7. Pull changes from remote: **git pull origin <branch>**
   Fetches changes from the remote branch and merges them into your current branch.
8. Create a new branch: **git branch <branch-name>**
   Creates a new branch from the current branch.
9. Switch branches: **git checkout <branch-name>**
   Moves to a different branch.

Advanced Git Concepts

1. Stashing: Temporarily save changes in the working directory. **git stash**
2. Rebase: Apply changes from one branch onto another, often used to integrate the latest changes from a main branch. **git rebase <branch>**
3. Cherry-pick: Apply a specific commit from one branch to another. **git cherry-pick <commit-hash>**
4. Revert: Undo a specific commit by creating a new commit. **git revert <commit-hash>**
5. Reset: Move the current branch to a specific commit, potentially discarding changes. **git reset --hard/--soft <commit-hash>**
6. Squash: Combine multiple commits into a single one, often used during a rebase. **git rebase -i <base-branch>**

Git Flow and Its usecases

The Git Flow tool is an extension to Git that provides a set of high-level commands to manage your branching strategy using the popular Git Flow methodology. Here's a breakdown of the main Git Flow commands and their usage.

1. **Setting Up Git Flow**

- Initialize Git Flow: Command: **git flow init**
- Description: Sets up your repository for Git Flow by creating the default branches (main and develop) and prompts you to set branch prefixes **(feature/, release/, hotfix/, etc.)**. This only needs to be done once per repository.

2. Working with Features

2.1 Start a Feature:

- Command: **git flow feature start <feature_name>**
- Description: Creates a new feature branch from develop to work on a specific feature. Naming it helps to identify what the feature does.

2.2 Finish a Feature:

- Command: **git flow feature finish <feature_name>**
- Description: Merges the completed feature branch back into develop and deletes the feature branch.

2.3 Publish a Feature:

- Command: **git flow feature publish <feature_name>**
- Description: Pushes the feature branch to a remote repository, allowing others to collaborate on it.

2.4 Pull a Feature:

- Command: **git flow feature pull <remote> <feature_name>**
- Description: Fetches and checks out a feature branch someone else has published to the remote.

Git Flow and Its usecases

3. Working with Releases

3.1 Start a Release:

- Command: **git flow release start <release_name> [<base>]**
- Description: Creates a new release branch from develop, preparing for a new production release. Useful for final bug fixes, version bumps, and other last-minute tasks.

3.2 Finish a Release:

- Command: **git flow release finish <release_name>**
- Description: Merges the release branch into both main and develop, tags the release on main with the release name, and deletes the release branch.

3.3 Publish a Release:

- Command: **git flow release publish <release_name>**
- Description: Pushes the release branch to a remote repository, allowing others to work on release fixes if needed.

4. Hotfixes

4.1 Start a Hotfix:

- Command: **git flow hotfix start <hotfix_name> [<base>]**
- Description: Creates a new hotfix branch from main to quickly address critical bugs in production

4.2 Finish a Hotfix:

- Command: **git flow hotfix finish <hotfix_name>**
- Description: Merges the hotfix branch into both main and develop, tags it on main with the hotfix name, and deletes the hotfix branch.

5. Supporting Commands

- List Git Flow Branches:
- Command: **git flow feature/release/hotfix list**
- Description: Lists all branches associated with features, releases, or hotfixes.