Priyanshu Dhyani

Git Basics Handbook

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 over time. By keeping track of every modification to the code in a special kind of database, a VCS allows developers to revert files to a previous state, compare changes over time, and work on different branches concurrently. This is crucial for collaborative projects where multiple developers are working on the same codebase.

B. Benefits of Utilizing Version Control for Software Development

- Collaboration: Multiple developers can work on the same project simultaneously without overwriting each other's work.
- History: VCS keeps a detailed history of changes, making it easy to track when and why a change was made.
- Backup: Acts as a backup mechanism, storing copies of all versions of the project.
- Branching and Merging: Facilitates the creation of branches for new features or bug fixes and merging them back into the main project seamlessly.
- Rollback: Easy to revert to previous versions of the project if new changes introduce bugs.

Core Concepts of Git

A. Repositories: Local and Remote

- Local Repository: This is the version of the repository on your local machine. It includes all the files and history of changes.
- Remote Repository: This is a version of the repository hosted on a server (like GitHub, GitLab, or Bitbucket). It's used for collaboration and sharing changes with other developers.

B. Working Directory: Workspace for Project Files

The working directory is where you modify your project files. It's essentially your local copy of the project where you make and test changes before committing them to the repository.

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

The staging area (or index) is a place where you can group changes you want to commit. It allows you to review changes and decide what to include in your next commit, ensuring that only complete and intentional updates are recorded.

D. Commits: Capturing Project States with Descriptive Messages

A commit is a snapshot of your repository at a specific point in time. Each commit has a unique ID and includes a descriptive message, making it easier to understand the history and purpose of changes.

Priyanshu Dhyani

E. Branches: Divergent Development Paths within a Repository

Branches are parallel versions of the repository. They allow developers to work on different features or fixes independently before merging their changes back into the main branch (usually called main or master).

Essential Git Commands

A. Initialization: Creating a New Git Repository

git init //Initializes a new Git repository in the current directory.

B. Tracking Changes: Identifying Modified Files

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

C. Staging and Committing: Preparing and Recording Changes

git add <file> //Stages the specified file(s) for commit.

git commit -m "Descriptive message" //Records the staged changes with a descriptive commit message.

D. Branching: Creating and Switching Between Development Lines

git branch <branch-name> //Creates a new branch.

git checkout <branch-name> //Switches to the specified branch.

E. Merging: Integrating Changes from Different Branches

git merge <branch-name> //Merges the specified branch into the current branch.

F. Remote Repositories: Collaboration and Shared Workspaces (Future Section)

git remote add origin <repository-URL> //Adds a remote repository.

git push origin <branch-name> //Pushes changes to the specified branch of the remote repository.

git pull origin <branch-name> //Pulls changes from the specified branch of the remote repository to your local branch.

Priyanshu Dhyani

Mastering Git Workflows

A. Feature Branch Workflow: Streamlined Development and Integration

- **Create a new branch:**
  git checkout -b feature-branch
- **Work on the feature:**
  Make changes and commit them.
- **Merge the feature branch:**
  Switch to the main branch and merge the feature branch.
  git checkout main
  git merge feature-branch

- **Delete the feature branch (optional):**
  git branch -d feature-branch

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

- **Create develop branch:**
  git checkout -b develop

- **Feature branches**: Branch off from develop for new features.
  git checkout -b feature-branch develop

- **Release branches:** When ready to release, create a release branch from develop.
  git checkout -b release-branch develop

- **Hotfix branches:** For urgent fixes, branch off from the main branch.
  git checkout -b hotfix-branch main

Advanced Git Techniques

A. Resolving Merge Conflicts: Handling Conflicting Changes

When merging branches, conflicts can occur if the same lines in the same files have been changed. Git will mark the conflicts in the files, and you'll need to resolve them manually before completing the merge.

git merge <branch-name> // If conflicts occur, edit the files to resolve them.

git add <resolved-file>

git commit

Priyanshu Dhyani

B. Stashing Changes: Temporarily Shelving Uncommitted Work

git stash //Temporarily saves your uncommitted changes and reverts your working directory to match the HEAD commit.

git stash pop //Restores the most recently stashed changes.

C. Using Tags: Annotating Specific Project Versions

git tag -a v1.0 -m "Version 1.0" //Creates an annotated tag named v1.0.

git push origin v1.0 //Pushes the tag to the remote repository.