Course Outline

Advanced Git Local Repository Maintenance

Course Summary

Description

This three-day course is designed for developers who are already familiar with Git and want to enhance their skills in managing local repositories. The course will cover advanced topics such as branching and merging strategies, rebasing, refactoring, and dealing with conflicts. The course will be hands-on, and participants will have the opportunity to work on real-world scenarios to reinforce their learning.

Objectives

At the end of this course, students will be able to:

- Understand branching and merging strategies, and choose the most appropriate one for their project needs.
- Refactor their Git repository to improve its structure and readability.
- Use rebase to keep their repository history clean and easy to understand.
- Resolve conflicts that may arise when merging branches.
- Use Git hooks to automate tasks and improve the development process.
- Debug and troubleshoot common Git issues.

Topics

- Branching and Merging
- Refactoring a Git Repository
- Rebase
- Dealing with Conflicts
- Git Hooks
- Troubleshooting Git

Audience

This course is designed for developers who are already familiar with Git and want to enhance their skills in managing local repositories.

Prerequisites

Foundational Git experience and knowledge are required including:

- core concepts of how git works
- creating and adding files to the local repository
- creating and navigating branches

Familiar with how to run git commands is assumed Comfortable working in bash (unix shell) is highly beneficial

Duration

Three days

Course Outline

Advanced Git Local Repository Maintenance

Course Outline

I. Branching and Merging

- A. Understanding different branching strategies
  1. Long-Running Branches
  2. Feature Branches
  3. Release Branches
  4. Hotfix Branches
- B. Choosing the appropriate strategy for the project
- C. Creating and managing branches
  1. Creating a new branch
  2. Switching branches
  3. Renaming branches
- D. Merging branches
  1. Fast-Forward Merges
  2. Merge Commits
  3. Merge Conflicts

II. Refactoring a Git Repository

- A. Identifying common repository issues
  1. Large files
  2. Unused files
  3. Inconsistent directory structure
- B. Techniques to refactor the repository
  1. Removing large files with Git LFS
  2. Using git clean to remove unused files
  3. Reorganizing directories with git mv
- C. Implementing refactoring changes
  1. Committing changes in small, atomic commits
  2. Using git bisect to identify problematic changes
- D. Testing the refactored repository
  1. Running tests to ensure changes didn't introduce bugs
  2. Using git blame to understand changes to the repository

III. Rebase

- A. Understanding the concept of rebasing
  1. Rebase vs. Merge
  2. Rebasing onto a different branch
- B. Benefits of using rebase
  1. Keeping a clean, linear history
  2. Avoiding merge commits
- C. Performing rebasing on the repository
  1. Using git rebase to rebase a branch
  2. Fixing conflicts during rebase
- D. Resolving conflicts during rebase
  1. Understanding how conflicts arise during rebase
  2. Resolving conflicts with git mergetool

IV. Dealing with Conflicts

- A. Understanding the cause of conflicts
  1. Conflicting changes to the same file
  2. Changes to the same lines in a file
- B. Identifying different types of conflicts
  1. Merge conflicts
  2. Rebase conflicts
- C. Strategies to resolve conflicts
  1. Using git diff to understand the conflict
  2. Manually editing the conflicting file
  3. Using git mergetool to resolve conflicts
- D. Testing the resolved repository
  1. Running tests to ensure changes didn't introduce bugs
  2. Using git bisect to identify problematic changes

V. Git Hooks

- A. Understanding Git Hooks
  1. Types of Git Hooks
  2. Writing Git Hooks in Bash or other scripting languages
- B. Setting up Git Hooks
  1. Configuring Git to use Hooks
  2. Writing a Hook script
- C. Common Git Hook use cases
  1. Pre-commit Hooks
  2. Post-commit Hooks
  3. Pre-push Hooks
- D. Debugging Git Hook issues
  1. Understanding why a Hook didn't run
  2. Testing a Hook script

VI. Troubleshooting Git

- A. Common Git issues and errors
  1. Repository Corruption
  2. Commit History Rewriting
  3. File Conflicts
- B. Debugging and resolving issues
  1. Using Git's built-in debugging tools
  2. Using third-party tools like GitKraken or Sourcetree
- C. Best practices for maintaining a healthy Git repository
  1. Regularly backing up the repository
  2. Keeping a clean commit history
  3. Educating team members on proper Git workflows.