Comprehensive Guide to Working with Commits in Git

Overview

Commits are the fundamental building blocks in Git that represent snapshots of your project. Understanding how to effectively manage commits can greatly improve your version control workflow.

Basic Commands

1. Initialize a Git Repository

Before creating commits, you need to initialize a Git repository:

git init

2. Add Files to the Staging Area

To include changes in the next commit, add files to the staging area:

git add <filename>

To add all changes:

git add .

3. Create a Commit

Create a commit with a meaningful message:

git commit -m "Your commit message"

Example:

git commit -m "Add README.md file"

Viewing Commit History

4. View Commit History

To see the history of commits in your repository:

git log

Tips:

1. To view a simplified, one-line summary of the commit history:

git log --oneline

1. To view a graphical representation:

git log --graph --oneline

Amending Commits

5. Amend the Most Recent Commit

If you need to modify the most recent commit message or include additional changes:

git commit --amend

Tips:

1. To amend only the commit message:

git commit --amend -m "New commit message"

Reverting and Resetting Commits

6. Revert a Commit

To create a new commit that undoes the changes of a previous commit:

git revert <commit_hash>

7. Reset to a Previous Commit

To reset your repository to a specific commit (use with caution):

git reset --hard <commit_hash>

Tips:

1. To keep the changes in your working directory while resetting:

git reset --soft <commit_hash>

Best Practices for Writing Commit Messages

8. Follow Conventional Commit Guidelines

A good commit message should be concise yet descriptive. Follow these guidelines:

1. **Subject line** (50 characters max): Summarize the changes.
2. **Body** (optional): Explain the changes, why they were made, and any additional context. Use bullet points for clarity.

Example:

feat: add user login feature

- Implement login form

- Validate user input

- Integrate authentication API

Advanced Features

9. Squash Commits

To combine multiple commits into one (useful for keeping history clean):

git rebase -i <base_commit>

Mark the commits you want to squash with `s` or `squash`.

10. Cherry-Pick Commits

To apply changes from a specific commit to the current branch:

git cherry-pick <commit_hash>

11. Signing Commits

To sign your commits cryptographically (ensure you have GPG set up):

git commit -S -m "Your commit message"

12. Bisecting

To find the commit that introduced a bug:

git bisect start

git bisect bad # Mark the current commit as bad

git bisect good <commit_hash> # Mark a known good commit

Git will help you narrow down the commit by checking out intermediate commits.

Tips and Tricks

13. Using Aliases

Create aliases for commonly used commands to save time:

git config --global alias.co checkout

git config --global alias.br branch

git config --global alias.ci commit

git config --global alias.st status

14. Stashing Changes

Temporarily save changes that you don't want to commit yet:

git stash

To apply stashed changes:

git stash apply

15. Viewing Differences

To see changes between commits or branches:

git diff <commit1> <commit2>

To compare with the staging area:

git diff --staged

16. Using Hooks

Automate tasks by using Git hooks (e.g., running tests before committing): Create a hook script in the `.git/hooks` directory. For example, a pre-commit hook:

#!/bin/sh

Run tests before commit

make test

Make the script executable:

chmod +x .git/hooks/pre-commit