GIT STARTER GUIDE

Introduction

In this guide, we will cover the basics of Git and how to get started with version control for your projects.

Welcome to the Git Starter's Guide for Students! Git is a powerful version control system that helps you manage and track changes in your code projects. Whether you are a computer science student or just getting started with programming, learning Git will be a valuable skill throughout your academic and professional journey. This guide will introduce you to the basics of Git and help you get started with version control.

Table of Contents

Chapter 1: What is Git and Version Control?

1.1 Understanding Version Control

Version control is a system that helps you keep track of changes in your code and collaborate with others effectively. It maintains a history of all modifications, allowing you to revert to previous versions if needed.

1.2 Introducing Git

Git is a distributed version control system that is widely used in the software industry. It allows multiple developers to work on a project simultaneously while keeping track of all changes.

Chapter 2: Installing Git and Initial Configuration

2.1 Installing Git

Install Git on your system by following the instructions for your operating system:

- Windows: Download the Git installer from the official website and follow the installation wizard.

- macOS: Git comes pre-installed on macOS. You can also install it using a package manager like Homebrew (brew install git).

- Linux: Use your package manager to install Git (e.g., sudo apt-get install git for Debian/Ubuntu).

2.2 Initial Configuration

After installing Git, set your username and email address using the following commands:

git config --global user.name "Your Name"

git config --global user.email "your.email@example.com"

Chapter 3: Basic Git Concepts

3.1 Git Repository

A Git repository is a directory where Git tracks your project's files and their changes. To initialize a new Git repository or clone an existing one, use the following commands:

git init # Initialize a new Git repository in the current directory

git clone <URL> # Clone an existing Git repository from a remote source

3.2 Committing Changes

Committing changes allows you to save your modifications to the codebase. Use the following commands to commit changes:

git status # View the status of your changes

git add <file> # Stage a specific file for the next commit

git add . # Stage all changes for the next commit

git commit -m "message" # Commit your changes with a meaningful message

3.3 Branching and Merging

Branching allows you to work on separate features or fixes without affecting the main codebase. Merge branches to combine changes. Use the following commands:

git branch # List all branches, showing the current branch with an asterisk (*)

git branch <branch_name> # Create a new branch

git checkout <branch_name> # Switch to an existing branch

git merge <branch_name> # Merge changes from the specified branch into the current branch

Chapter 4: Collaborating with Git

4.1 Cloning a Repository

To collaborate on a project, clone an existing remote repository to your local machine using the following command:

git clone <URL> # Clone a remote repository to your local machine

4.2 Pulling Changes

Keep your local repository up to date with the latest changes from the remote repository using the following command:

git pull origin <branch> # Pull the latest changes from the remote repository into your current branch

4.3 Pushing Changes

Upload your local changes to the remote repository to share your work with others. Use the following command:

git push origin <branch> # Push your local changes to the remote repository in the specified branch

4.4 Handling Merge Conflicts

When merging branches, conflicts may arise if Git can't automatically resolve differences. Manually resolve conflicts using the following steps:

1. After attempting to merge, Git will indicate which files have conflicts.

2. Open the files and manually edit them to resolve conflicts.

3. Once conflicts are resolved, add and commit the changes.

git add <file> # Stage the resolved changes

git commit -m "message" # Commit the resolved changes

Chapter 5: Advanced Git Topics (Optional)

5.1 Git Ignore

Create a .gitignore file to specify which files and directories Git should ignore. Add patterns for ignored files and directories like this:

Create a new .gitignore file using a text editor and add patterns for ignored files and directories.

For example, to ignore all .log files and the 'node_modules' directory:

*.log

node_modules/

5.2 Undoing Changes

Undo commits and revert changes using various Git commands like reset and revert (use with caution).

git log # View the commit history to identify the commit you want to undo

git reset --hard <commit_hash> # Move the branch pointer to the specified commit, discarding subsequent changes

git revert <commit_hash> # Create a new commit that undoes the changes introduced by the specified commit

5.3 Git Workflow Strategies:

Explore different Git workflows like Gitflow and GitHub Flow to understand more advanced collaboration techniques.

Conclusion:

Congratulations! You have completed the Git Starter's Guide for Students. By mastering Git, you've gained a fundamental skill that will help you collaborate effectively on coding projects and enhance your software development capabilities. Remember to keep practicing and exploring more advanced Git features as you progress in your programming journey. Happy coding!