Git Tutorial: A Comprehensive Guide

Introduction to Git

Git is a distributed version control system designed to handle everything from small to very large projects with speed and efficiency. It helps developers track changes in their code, collaborate with others, and maintain a history of their work.

Key Concepts

- **Repository**: A directory that contains your project files and the history of their changes.
- **Commit**: A snapshot of your repository at a specific point in time.
- **Branch**: A parallel version of your repository. It's contained within the repository but does not affect the main branch.
- **Merge**: Combining changes from different branches.

Installation

Windows

1. Download the installer from the Git website.
2. Run the installer and follow the setup instructions.

macOS

You can install Git using Homebrew:

```
brew install git
```

Linux

For Debian-based systems:

```
sudo apt-get update
sudo apt-get install git
```

For Red Hat-based systems:

```
sudo yum install git
```

Basic Configuration

After installing Git, configure your username and email. This information will be associated with your commits.

```
git config --global user.name "Your Name"
git config --global user.email "you@example.com"
```

You can check your configuration with:

```
git config --list
```

Creating a New Repository

To create a new repository, navigate to the directory where you want the repository to live and run:

```
git init my-repo
cd my-repo
```

This creates a new directory called `my-repo` and initializes an empty Git repository.

Cloning an Existing Repository

To clone a repository from a remote source (like GitHub):

```
git clone https://github.com/username/repo.git
```

This command copies all the files and the commit history from the specified repository.

Basic Git Commands

1. Checking the Status

To check the status of your repository, use:

```
git status
```

This shows you the current branch, staged changes, and any untracked files.

2. Adding Changes

To stage files for commit, use:

ASHU

```
git add filename
```

To stage all changes:

```
git add .
```

3. Committing Changes

After staging your changes, commit them with a message:

```
git commit -m "Your commit message"
```

4. Viewing Commit History

To view the commit history, use:

```
git log
```

You can see a list of commits along with their IDs and messages.

5. Creating a Branch

To create a new branch:

```
git branch new-branch
```

6. Switching Branches

To switch to a different branch:

```
git checkout new-branch
```

You can combine creating and switching in one command:

```
git checkout -b new-branch
```

7. Merging Branches

To merge changes from another branch into your current branch:

```
git merge new-branch
```

8. Deleting a Branch

To delete a branch that you no longer need:

ASHU

```
git branch -d branch-name
```

Working with Remote Repositories

Adding a Remote Repository

To add a remote repository:

```
git remote add origin https://github.com/username/repo.git
```

Pushing Changes

To push your commits to a remote repository:

```
git push origin main
```

Pulling Changes

To pull changes from the remote repository:

```
git pull origin main
```

Fetching Changes

To fetch changes from the remote without merging:

```
git fetch origin
```

Resolving Merge Conflicts

If two branches have changes in the same lines of a file, you may encounter a merge conflict. Git will mark the conflict in the file, and you will need to manually resolve it.

1. Open the file and look for conflict markers (`<<<<<<<`, `=======`, `>>>>>>>`).
2. Edit the file to resolve the conflict.
3. After resolving, stage the changes:
4. `git add filename`
5. Finally, commit the changes:
6. `git commit -m "Resolved merge conflict"`

Undoing Changes

Unstaging a File

ASHU

If you want to unstage a file:

```
git reset HEAD filename
```

Discarding Changes

To discard changes in a file:

```
git checkout -- filename
```

Reverting a Commit

To revert a specific commit (create a new commit that undoes the changes):

```
git revert commit-id
```

Conclusion

Git is an essential tool for developers, enabling efficient version control and collaboration. This tutorial covered the basics to get you started. As you grow more comfortable with Git, you can explore advanced features such as rebasing, stashing, and using hooks.

Happy coding!

ASHU