Timeless,

GIT Concepts,

To Sharpen Your Skills !

@dinukanilupul

Next »

Motivation

The most-used technology by developers isn’t any programming language at all; it’s Git and version control, although it rarely gets mentioned in job requirements or interviews. Many of us get by learning only basic Git concepts to work in small teams, but this minimal knowledge can only take us so far. Mastering Git can significantly improve both your coding workflow and career progression. This guide focuses on essential but often overlooked Git concepts, aimed at those already familiar with the basics. If you’re comfortable with general Git commands and concepts, this guide will give you a boost for your knowledge.

1. Git Log

The git log command is essential for viewing commit history, allowing you to track changes over time and understand who made changes and why. It's crucial for debugging and reviewing project progression.

```
git log --oneline --graph --all
```

Displays each commit in a single line, making it easier to scan history.

Adds a visual branch structure to the history, helping you see how commits relate to each other.

Shows all branches, not just the current one, so you can view the entire project's history.

Next »

2. Git Reflog

git reflog allows you to track changes made to references (like HEAD). This is a safety net for restoring changes you thought were lost, such as after a reset.

```
git reflog
```

This, lists every action that moved the HEAD reference, including commits, resets, and checkouts, letting you recover from mistakes by reverting to previous states.

3. Git Branching

Branches allow developers to work on different features or fixes independently, which is vital for team collaboration and clean project structure.

```
git branch new-feature
```

Creates a new branch called new-feature without switching to it.

```
git checkout -b new-feature
```

This command to create new branch and switch to it. If branch is already exist, this won't create new branch!

```
git checkout new-feature
```

```
git switch new-feature
```

Switches your working directory to the new-feature branch, isolating your work from other branches.

4. Undoing Changes with Reset and Restore

git reset and git restore are useful for discarding changes. reset undoes commits, and restore lets you revert changes in your working directory or staging area.

```
git reset --soft HEAD~1
```

Moves the HEAD back by one commit without touching the working directory, so your changes remain staged.

```
git restore <file-name>
```

Reverts changes made to a specific file in the working directory without affecting other files.

5. Amend Commits

git commit --amend lets you edit the most recent commit, perfect for making quick fixes or adjusting messages before sharing changes.

```
git commit --amend -m "Updated commit message"
```

Replaces the message of the latest commit with the new one provided, or updates the contents if other changes are staged.

6. Merge vs. Rebase

Both merge and rebase integrate changes from one branch to another. Merging preserves branch history, while rebasing rewrites it for a cleaner, linear history.

```
git merge feature-branch
```

Integrates feature-branch into the current branch, preserving history.

```
git rebase main
```

Moves the current branch onto main, rewriting commit history for a linear look. This makes the commit history tree looks clean. But make it difficult to track changes.

7. Cherry-Picking

git cherry-pick allows you to apply a specific commit from one branch to another. This is useful for pulling a bug fix into a different branch without merging everything else.

```
git cherry-pick <commit-hash>
```

<commit-hash>: it's the unique identifier of the commit you want to copy. cherry-pick applies this exact commit to your current branch.

8. Stashing

git stash temporarily saves uncommitted changes, allowing you to switch branches without losing progress. Useful when you need to handle something urgent on another branch. It works like, picking changes and keep it safe in separate to apply later.

```
git stash
```

Saves current changes without committing them, clearing your working directory.

```
git stash pop
```

Retrieves the last stashed changes and applies them back to the working directory.

9. Remote Operations

Remote commands like push, pull, and fetch allow you to synchronize with a remote repository, essential for team collaboration.

```
git push origin main
```

Uploads local main branch commits to the origin remote repository.

```
git fetch origin
```

Retrieves updates from origin without merging them.

```
git push origin main
```

Fetches and merges changes from the main branch of origin into your local branch.

10. Aliases for Efficiency

Setting aliases for common commands can save time and improve efficiency, especially with frequently used commands.

```
git config --global alias.co checkout
```

Sets git co as a shortcut for git checkout.

```
git config --global alias.br branch
```

Sets git br as a shortcut for git branch, making it faster to navigate and manage branches.

Next »

Follow For More !

Learn Together, Grow Together

Dinuka Nilupul

@dinukanilupul