Master Git: Understanding Key Commands for Effective Version Control

Fetch, Pull, Revert, Reset, Rebase, and Merge Explained

MAMA SAMBA BRAIMA NELSON DJALO

JUN 27, 2024

Git is an essential tool for developers, enabling efficient version control and collaboration. However, with its vast array of commands, it can be overwhelming for beginners to grasp their usage and implications fully.

This blog post aims to demystify six crucial Git commands by categorizing and explaining their functionalities.

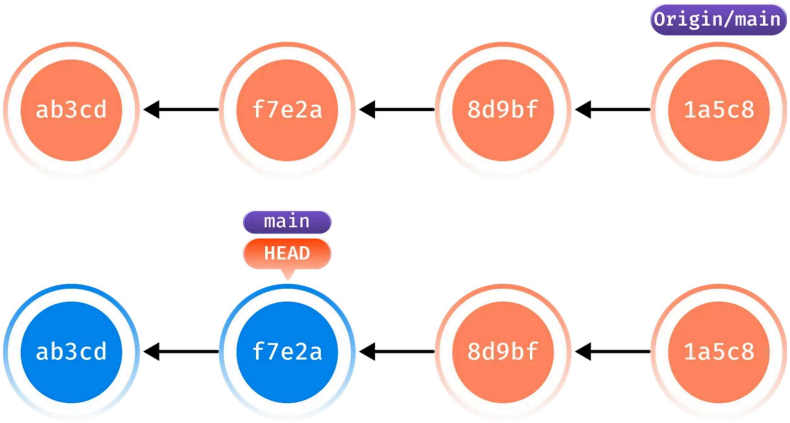
- fetch
- pull
- revert
- reset
- rebase
- merge

Now lets dive into each command.

Please note some still images really make more sense with animation.

Fetching and Synchronizing

Fetch

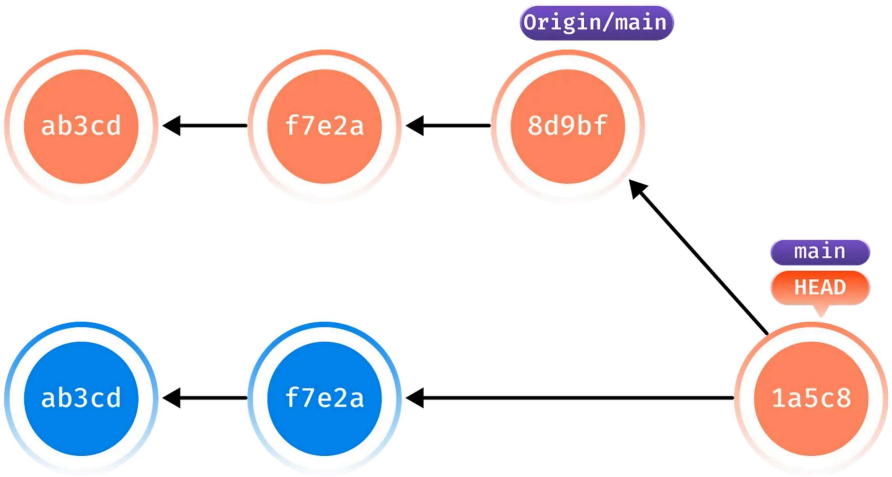


The `fetch` command downloads objects and refs from another repository. It updates your remote-tracking branches but doesn't merge these changes into your working files. This allows you to review changes before integrating them into your project.

Usage:

```
git fetch <remote>
```

Pull



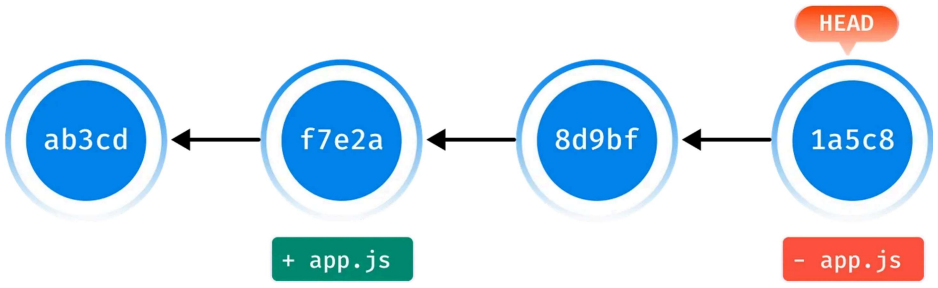
The `pull` command is essentially a combination of `fetch` and `merge`. It downloads objects and refs from another repository and immediately merges them into your current branch. This is a more straightforward approach but doesn't offer the same level of control as using `fetch` followed by a manual merge.

Usage:

```
git pull <remote> <branch>
```

Undoing Changes

Revert



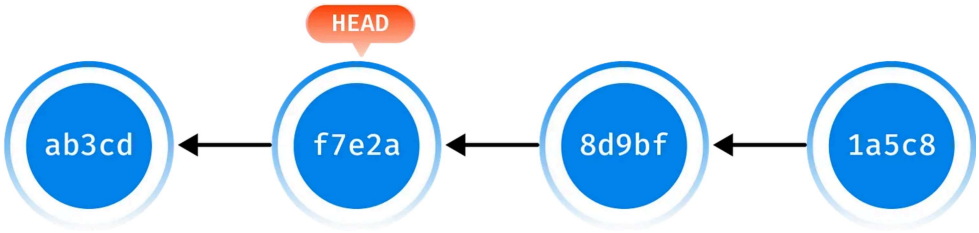
The `revert` command creates a new commit that undoes the changes made by a previous commit. It's useful for reversing changes in a project while keeping a record of

these reversals.

Usage:

```
git revert <commit>
```

Reset



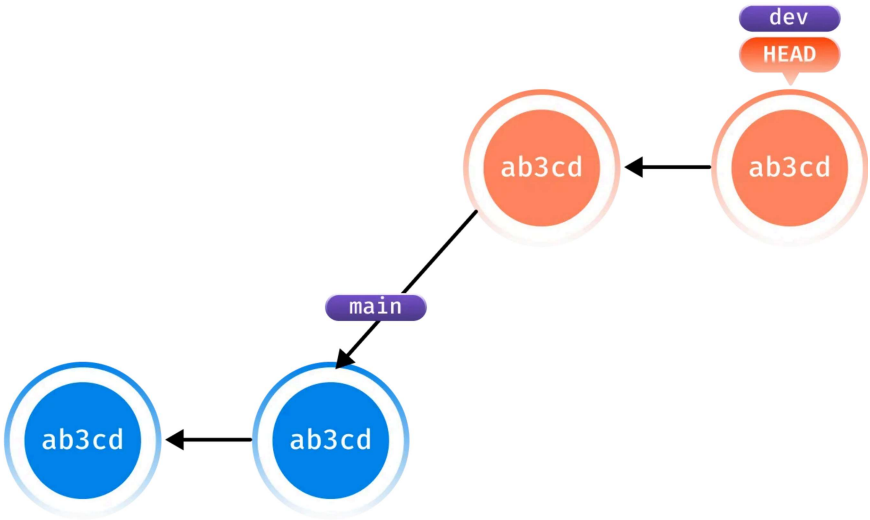
The `reset` command is used to move the current branch to a different commit. Depending on the mode used (`--soft`, `--mixed`, `--hard`), it can also modify the index and the working directory to match the specified commit. This command is powerful but can be dangerous as it can rewrite history.

Usage:

```
git reset --soft|--mixed|--hard <commit>
```

Manipulating History

Rebase



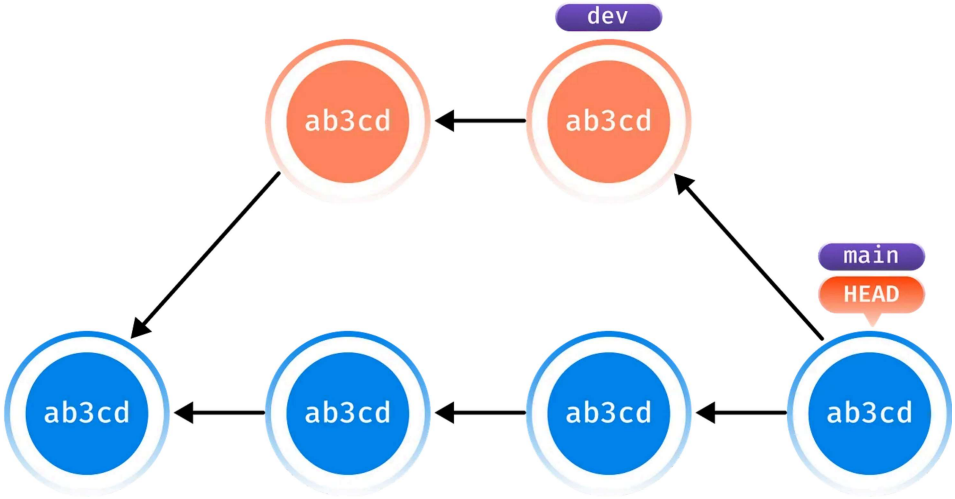
The rebase command is used to reapply commits on top of another base tip. It's a powerful tool for maintaining a linear project history by moving or combining a sequence of commits to a new base commit.

Usage:

```
git rebase <base>
```

Integrating Changes

Merge

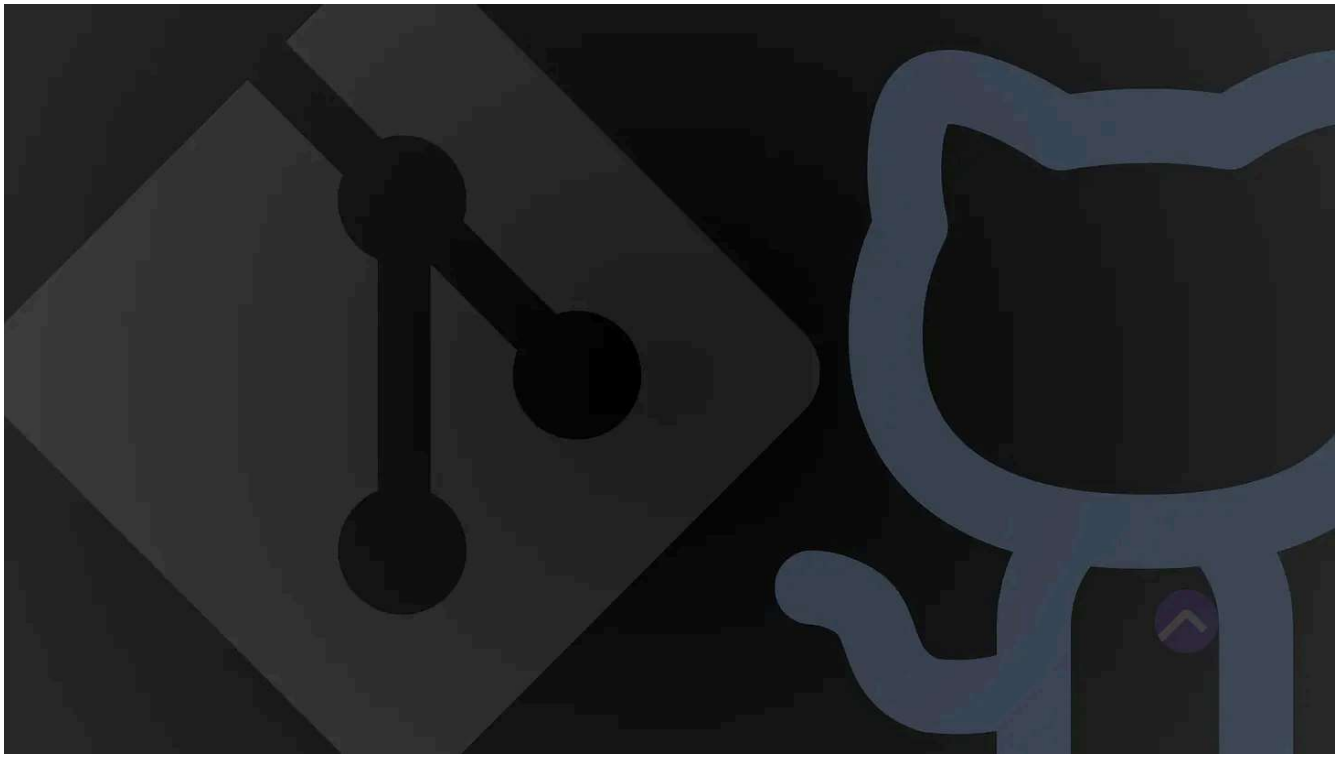


The `merge` command joins two or more development histories together. It takes the contents of a source branch and integrates them with the current branch, creating a new merge commit in the process.

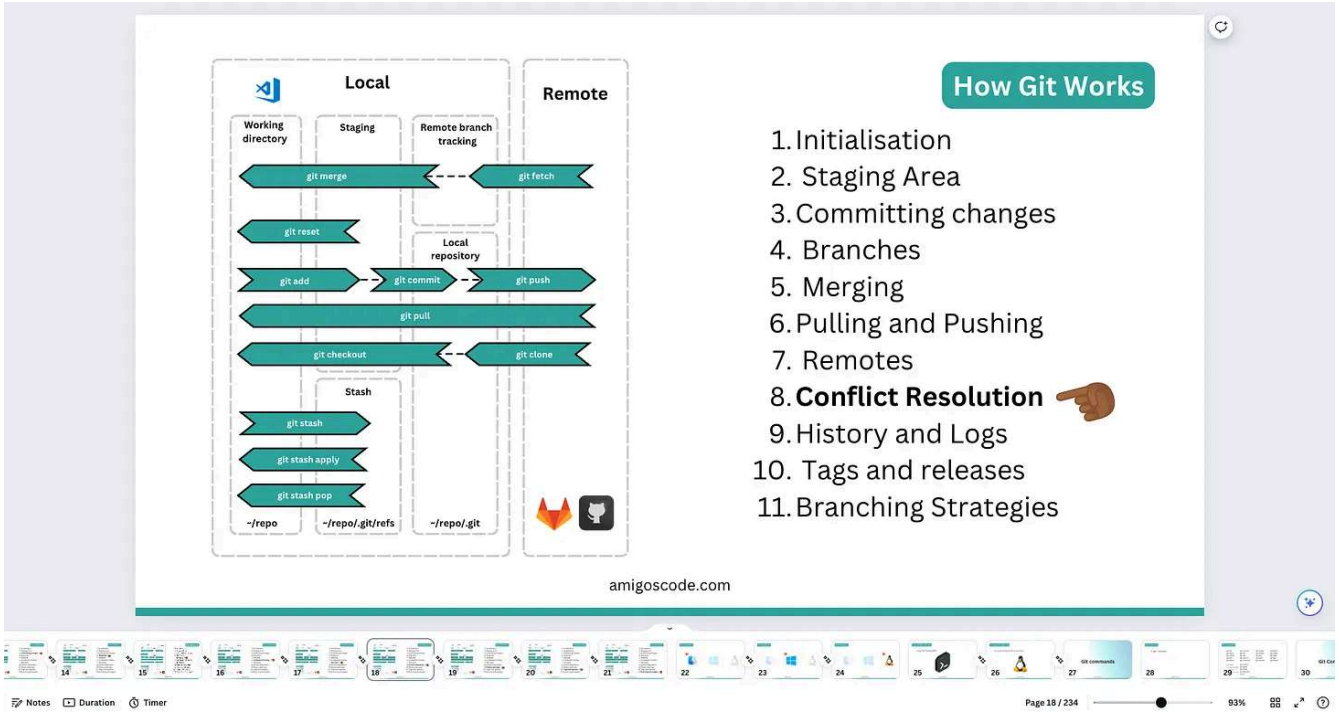
Usage:

```
git merge <branch>
```

Join waiting list for our 15 hours git course which covers all of these topics and more 🎉



Including over 230 Slides With Clear Explanations



Here is what your learn

- Introduction To Git
- Anatomy of Git Commands
- Creating Your First Git Repository
- Tracking Changes In Git
- Working With Remote Repositories
- Practical Exercises
- Excluding Files And Folders In Git
- Taking A Closer Look At Commits
- Undoing Changes In Git
- Force Pushing In Git
- Working With Branches
- Synchronizing Changes With Remote Repositories
- Working With Pull Requests
- Collaborating On Pull Requests
- Managing Merge Commits
- Rebasing In Git
- Squashing Commits In Git
- Resolving Merge Conflicts
- Real-World Examples Of Rebase And Conflicts
- Using Git Stash
- Advanced Git Techniques And Best Practices
- Using Visual Studio Code With Git
- Git Clients (VSCode, IntelliJ and Others)

- Open Source And Collaborative Workflows
- Markdown And Documentation
- Using Codespaces
- Continuous Integration And Deployment (CI/CD)
- Enhancing Security In Git
- Agile Project Management And Planning
- AI-Powered Development With GitHub Copilot

See you on the next post 🤝

25 Likes

← Previous

Comments

Write a comment...