Advanced Git

Interactive Rebase

A tool for optimizing and cleaning up your commit history. You can

- Change a commit's message
- Delete commits
- Reorder commits
- Combine multiple commits into one
- Edit / split an existing commit into multiple new ones

DO NOT use Interactive Rebase on commits that you've already pushed / shared on a remote repository

Instead, use it for cleaning up your local commit history before merging it into a shared team branch.

Workflow

Step 1: How far back do you want to go? What should be the "base" commit?

Step 2: $ git rebase -i HEAD~3

Step 3: In the editor, only determine which actions you want to perform. Don't change commit data in this step, yet.

Step 4: A new editor will pop up and will let you do the action that you chose.

Cherry Picking

Integrating Single, Specific Commits.

Cherry-pick allows you to select individual commits to be integrated using their commit hash.

Workflow

Step 1: Checkout the branch where you want to integrate those specific commits

Step 2: git cherry-pick <commit_hash>

Reflog

A protocol of HEAD pointer movements.

Consider a situation:

1. You think you want to get rid of some commits
2. You use "git reset"
3. After 5 minutes, you notice that it was a bad idea
4. Panik

Recovering Deleted Commits – Workflow

Step 1: git reflog

Step 2: Check which state of the repo you want to get to

Step 3: git branch <some_new_branch> <commit_hash_of_the_state>

Consider one more situation:

1. You think you don't need a feature branch anymore
2. You delete it
3. After 5 minutes, you notice that it was a bad idea
4. Panik

Recovering Deleted Branches – Workflow

Step 1: git reflog

Step 2: Check which state of the repo you want to get to

Step 3: git branch <some_new_branch> <commit_hash_of_state>

Submodules

Consider a situation where you want some external library / third-party code in your project. So, you download that file and copy and paste it in your project repo. The downsides of doing this is that you are mixing external code with your own files and later in time if you want to update that external code, you will have to manually do that. This is small brain.

A **Submodule** is a git repository inside a git repository.

Workflow:

Step 1: Create a "lib" folder or something so that the external code is neatly separated from our code

Step 2: git submodule add <remote_repo_url>

This will create a ".gitmodules" file in our repository.

Step 3: Carry on with your project

Search and Find

Filtering your commit history. You can filter:

1. By date --before / --after
2. By message --grep
3. By author --author
4. By file --<filename>

5. By branch <branch-A>

Search log after a particular date

git log –-after=”2021-8-2” (show all logs after August 2nd)

Search for a particular pattern in the logs

git log –-grep=”refactor”

Search for a log by a certain author

git log --author=”Keanu Reeves”

Search logs by filename

git log -- README.md

Search logs by branch

git log feature/login..main (shows all logs which are in main but not in feature branch)