Git Commits

A commit in a git repository records a **snapshot** of all the (tracked) files in your directory.

Git wants to keep commits as **lightweight** as possible though, so it doesn't just blindly copy the entire directory every time you commit. It can (when possible) compress a commit as a set of changes, or a "**delta**", from one version of the repository to the next.

Git also maintains a history of which commits were made when. That's why most commits have ancestor commits above them

Git Branches

Branches in Git are incredibly lightweight. They are simply pointers to a specific commit.

There is no extra storage / memory overhead with making many branches,

it's easier to logically divide up your work than have big beefy branches.

Git Merging

Merging in Git creates a special commit that has two unique parents.

Git Rebase

The second way of combining work between branches is rebasing. Rebasing essentially takes a set of commits, "copies" them, and plops them down somewhere else.

The advantage of rebasing is that it can be used to make a nice linear sequence of commits. The commit log / history of the repository will be a lot cleaner if only rebasing is allowed.

HEAD

HEAD is the symbolic name for the currently checked out commit, it's essentially what commit you're working on top of.

HEAD always points to the most recent commit which is reflected in the working tree. Most git commands which make changes to the working tree will start by changing HEAD.

Detaching HEAD

Detaching HEAD just means attaching it to a commit instead of a branch.

*git checkout <commit sha>*

Relative Refs

Moving upwards one commit at a time with *HEAD^ | HEAD~* (the first parent of a commit)

Moving upwards two commits at a time with *HEAD^^* (grandparent of a commit)

Moving upwards one commit at a time with *HEAD^2* (second parent of a commit) | (Used in merged commits)

Moving upwards a number of times with *HEAD ~<num>*

Use relative refs is to move branches around. You can directly reassign a branch to a commit with the -f option.

*git branch -f main HEAD~3*

moves (by force) the main branch to three parents behind HEAD.

A screenshot of a computer

Description automatically generated with medium confidenceA screenshot of a computer

Description automatically generated with medium confidence

Git Reset and Git Revert

git reset reverses changes by moving a branch reference backwards in time to an older commit. In this sense you can think of it as "rewriting history;" git reset will move a branch backwards as if the commit had never been made in the first place.

While resetting works great for local branches on your own machine, its method of "rewriting history" doesn't work for remote branches that others are using.

*git reset HEAD~1*

In order to reverse changes and share those reversed changes with others, we need to use git revert.

*git revert HEAD*

Git Cherry-pick

Copy a series of commits below your current location (HEAD).

*git cherry-pick <Commit1> <Commit2> <...>*

Git Interactive Rebase

All interactive rebase means is using the rebase command with the -i option.

If you include this option, git will open up a UI to show you which commits are about to be copied below the target of the rebase. It also shows their commit hashes and messages, which is great for getting a bearing on what's what.

*git rebase -i HEAD~4 (take previous 4 commits in interactive session & create a new linear structure)*

Git Tags

They (somewhat) permanently mark certain commits as "milestones" that you can then reference like a branch.

More importantly though, they never move as more commits are created. You can't "check out" a tag and then complete work on that tag. Tags exist as anchors in the commit tree that designate certain spots.

*Reference Link : https://learngitbranching.js.org/*