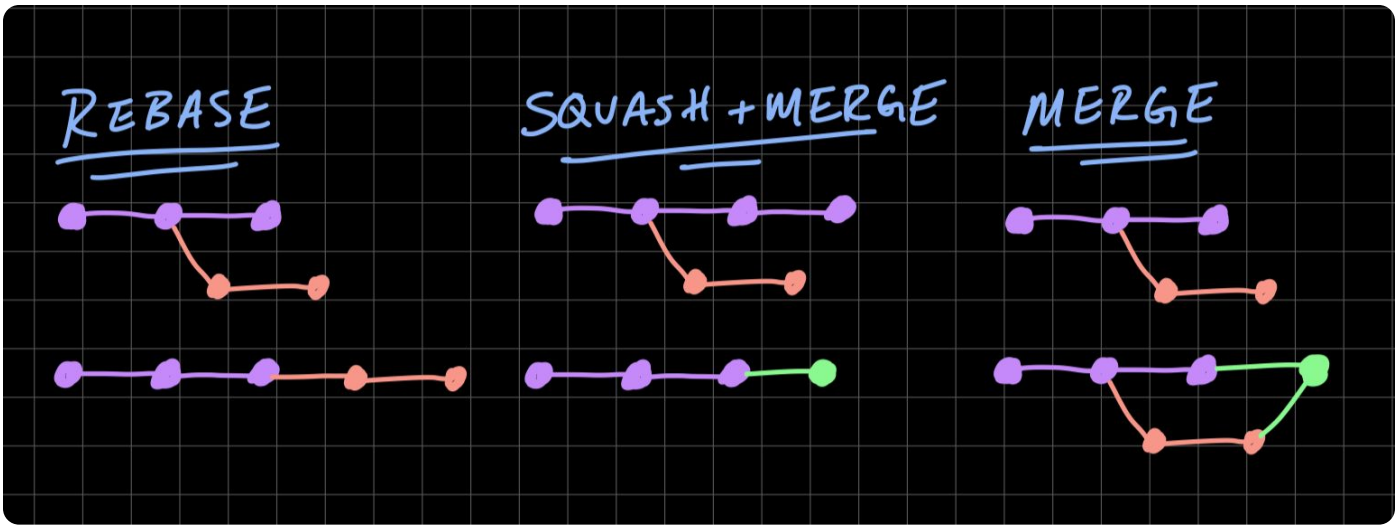
Squash, Merge, or Rebase?

Feb 25, 2022



When version controlling your code with git, there are generally three choices when merging feature branches into main. Each has its quirks, so which one should you use?

Rebase rewrites history on top of a branch. This provides a linear history, meaning context is lost of where a feature branched off. You may also have to force push changes (since you are rewriting history) if you have already pushed to a remote.

Merge will create a merge commit that joins two branches together. With the fast-forward-only flag **`ff-only`**, git will attempt to merge without a merge commit, but this isn't possible if the branches have diverged (i.e., there has been a commit to the parent branch that's not on the feature branch).

Squash + Merge acts like merge but creates a single new squashed commit that encompasses all commits in the feature branch.

I use rebase. Rebase retains a linear commit history, which is important for rollbacks. In addition, rebase is the most flexible workflow – larger and more difficult merges can be tough to do with a squash and merge. Interactive rebase has a bit of a steeper learning curve, but with practice, it can work in all scenarios. Squash and merge is OK for smaller changes, but it might be smarter to break large feature branches into multiple logical commits. This is useful for cherry-picking commits to other branches or repositories.