#2 Intro to Changing History

#advanced-git

In general, it's a bad idea to change **established** history. The exact definition of "established" depends on the context, but in general, any history that is visible to people other than you is considered established. A common example is any commits that have been pushed to a git remote. It's slightly less bad to change history that is established if you're the only person that will be changing it.

With that out of the way, let's talk about *why* you might even want to change history. Some possible reasons:

> A quick summary of the information below:
> 1. Fixing a small mistake — amend the commit.
> 2. Undoing old changes — use revert to create a new commit.
> 3. Keeping a branch up to date — rebase onto the source branch.
> 4. Cleaning up history, while preserving changes — interactive rebase.
> 5. Removing commits — reset.

1. **Fixing a small mistake.** "Small" of course depends on the context, but let's say that it affects one commit. The easiest way to fix this is by using `git commit --amend`.
2. **Undoing changes introduced by old commits.** The best way of doing this is by creating a *new* commit that does the opposite of those old commits. The can be done using `git revert`. Note that this actually will not change history.
3. **Updating a stale branch.** In my opinion, this is a good habit to get into for any feature work that occurs separate from a parent branch. This also has the side benefit of a cleaner commit history as well as making it easier to merge later. This can be done using `git rebase`.
4. **Combining old commits in a complicated history.** This is a good strategy for simplifying unnecessarily complicated history. This can be achieved by a combination of `git reset` and making new commits, but the most streamlined way of doing this is through `git rebase --interactive`.
5. **Getting rid of commits for good.** This is dangerous and can be very difficult to undo. It can be accomplished using `git reset`.