Changing The Last Commit

You've already made plenty of commits with the `git commit` command. Now with the `--amend` flag, you can alter the *most-recent* commit.

```
$ git commit --amend
```

If your Working Directory is clean (meaning there aren't any uncommitted changes in the repository), then running `git commit --amend` will let you provide a new commit message. Your code editor will open up and display the original commit message. Just fix a misspelling or completely reword it! Then save it and close the editor to lock in the new commit message.

Add Forgotten Files To Commit

Alternatively, `git commit --amend` will let you include files (or changes to files) you might've forgotten to include. Let's say you've updated the color of all navigation links across your entire website. You committed that change and thought you were done. But then you discovered that a special nav link buried deep on a page doesn't have the new color. You *could* just make a new commit that updates the color for that one link, but that would have two back-to-back commits that do practically the exact same thing (change link colors).

Instead, you can amend the last commit (the one that updated the color of all of the other links) to include this forgotten one. To do get the forgotten link included, just:

- edit the file(s)
- save the file(s)
- stage the file(s)
- and run `git commit --amend`

So you'd make changes to the necessary CSS and/or HTML files to get the forgotten link styled correctly, then you'd save all of the files that were modified, then you'd use `git add` to stage all of the modified files (just as if you were going to make a new commit!), but then you'd run `git commit --amend` to update the most-recent commit instead of creating a new one.