Lesson | Wednesday

Introduction to Programming (/introduction-to-programming) / Arrays and Looping (/introduction-to-programming/arrays-and-looping) / Rewriting Git History with Rebase

Text | Cheat sheet

We always want to include clear and detailed messages with our commits. However, even the most careful developers can make mistakes. Thankfully, Git offers several options to re-write commits.

But first, note that we need to be very careful when modifying Git history. This is especially true if you are working collaboratively with others: If you modify an existing commit, it actually *removes* the old commit from the project and makes a *new* one in its place. If others are working on the same branch they will still have your old commits when they pull down any changes, and will be asked to merge your *rewritten* commits with your *new* commits. This can create a real mess. For that reason, **you should not modify commit history on a branch others are also working on**.

However, in some circumstances it is necessary to modify your **local** commit history (that is, commits that exist locally on your machine, but have not been pushed to Github yet) before pushing your repository. In this lesson, we'll cover several ways to make changes to your Git commit history.

`$ git log --oneline`

We can see a project's commit history with the `git log` command. And we can add the `--oneline` flag to display the history in an easier to read format:

```
$ git log --oneline
d7e33de something css related
32ccc0b do something with html
9db82b6 update readme
79e9de2 add readme
827ad58 add initial files
```

We can see that this project contains some poorly-worded commits. There are also two consecutive commits about the same feature. Let's clean up this Git commit history!

`$ git commit --amend`

First we'll learn how to modify the **most recent** commit. Let's say we committed the right files, but we messed up the commit message. In this case we can use `git commit --amend` to simply update the commit message:

```
$ git commit --amend -m "add styling to main page"
```

This changes the commit message on the most recent commit from "something css related" (as seen in the git log above) to "add styling to main page". Note that it also assigns that commit a new id, since it deleted the old commit and created a new one in its place. The Git history reflects the change:

```
$ git log --oneline
6ad9b62 add styling to main page
32ccc0b do something with html
9db82b6 update readme
79e9de2 add readme
827ad58 add initial files
```

If we accidentally forgot to include a file in the previous commit, we could also use `git commit --amend` to add a file to the previous commit:

```
$ git add index.html
$ git commit --amend -m "add styling to main page"
```

`$ git rebase -i`

`git commit --amend` is an easy way to modify our most recent commit, but if we need to modify history going *further* back then we'll need to use the powerful (but potentially dangerous!) `git rebase -i` command.

`git rebase -i` allows us to change commit messages and combine multiple commits by "squashing" them together. Be particularly careful with this command. It permanently deletes all commits from the point you're modifying onward, replacing them with new commits.

Changing old commit messages with `$ git rebase -i`

Now, let's reword our second-to-last commit's message. If we type `git rebase -i HEAD~2` it will launch the system editor, where we'll see a Git rebase file containing the two most recent commits. The first commit listed will be the commit that we requested to rebase,

and any subsequent commit will be more and more recent in history up until the last commit we made. It will look something like this:

```
pick 32ccc0b do something with html
pick 6ad9b62 add styling to main page
```

To reword some of the commit messages, we can change `pick` before those lines to `reword`, as follows:

```
reword 32ccc0b do something with html
pick 6ad9b62 add styling to main page
```

When we save and close that file, we're immediately presented with a commit message file in the editor, allowing us to update the commit message we marked for rewording:

```
do something with html

# Please enter the commit message for your changes. Lines s
tarting
# with '#' will be ignored, and an empty message aborts the
commit.
#
# Date:      Sat Mar 19 10:56:31 2016 -0700
#
# rebase in progress; onto 9db82b6
# You are currently editing a commit while rebasing branch
'main' on '9db82b6'.
#
# Changes to be committed:
#       modified:   index.html
#
```

Let's change the commit message at the top to "add welcome message to index.html". We save and close the file, and now the changes are reflected in the Git history:

```
$ git log --oneline
0e3e0bc add styling to main page
1a940bb add welcome message to index.html
9db82b6 update readme
79e9de2 add readme
827ad58 add initial files
```

Note that the last two commits (the two we rebased) now have new id's, because the old commits were deleted and replaced with new ones.

Combining Multiple Commits with `$ git rebase -i`

Now, let's say we want to combine multiple commits for one feature into a single commit before publishing to Github. We want to combine the "add readme" and "update readme" commits.

We can type `git rebase -i HEAD~4` to bring up the four most recent commits in the rebase editor window. Again, this list will start with the commit that we requested to rebase from, the fourth commit going back into our Git history.

To combine a commit with the previous commit, we can change `pick` to `squash`.

Let's change `pick` to `squash` on the "update readme" commit in order to combine that one with the "add readme" commit directly preceding it:

```
pick 79e9de2 add readme
squash 9db82b6 update readme
pick 1a940bb add welcome message to index.html
pick 0e3e0bc add styling to main page
```

After saving and closing this file, we're presented with a commit message file. We can write the commit message for the new *combined* commit in this file:

```
# This is a combination of 2 commits.
# The first commit's message is:

add readme

# This is the 2nd commit message:

update readme
```

We can update the first commit message to "create project readme" and simply delete the "update readme" line, since we're eliminating the second commit from our project:

```
# This is a combination of 2 commits.
# The first commit's message is:

create project readme

# This is the 2nd commit message:
```

After saving and closing that file, the Git history reflects the change:

```
$ git log --oneline
59e7083 add styling to main page
2b1b367 add welcome message to index.html
4c876ed create project readme
827ad58 add initial files
```

Pushing Rewritten History to Github

Again, be extremely careful when changing history on a public repository. Do not make history changes to a remote repository being worked on by other developers. However, you may push rewritten history to a public repository if you're the *only* developer working on it.

Normally you'll receive the following error if you push a project to Github, rewrite history, and then attempt to push the project again:

```
$ git push origin main
To https://github.com/epicodus-lessons/hello
 ! [rejected]        main -> main (non-fast-forward)
error: failed to push some refs to 'https://github.com/epic
odus-lessons/hello'
hint: Updates were rejected because the tip of your current
branch is behind
hint: its remote counterpart. Integrate the remote changes
(e.g.
hint: 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --hel
p' for details.
```

This is simply because the local commits and remote commits no longer match. In order to force Github to throw away the old version of the project and replace it with the newly-edited local commit history, you must use the `--force` option:

```
$ git push origin main --force
```

Rebasing to Clean Merge

Note that `git rebase` without the `-i` flag is often used for doing a "clean merge" of different branches. We won't cover this concept yet, simply know this option exists in case you encounter it in the future.

Or, if you'd like to explore it now, you can find more information by reading this Atlassian article on Git Rebase (https://www.atlassian.com/git/tutorials/rewriting-history/git-rebase).

Previous (/introduction-to-programming/arrays-and-looping/practice-using-tdd-with-text-analyzer)
Next (/introduction-to-programming/arrays-and-looping/practice-rewriting-git-history-with-rebase)

Lesson 33 of 50
Last updated March 24, 2023

disable dark mode

(http://www.epicodus.com)