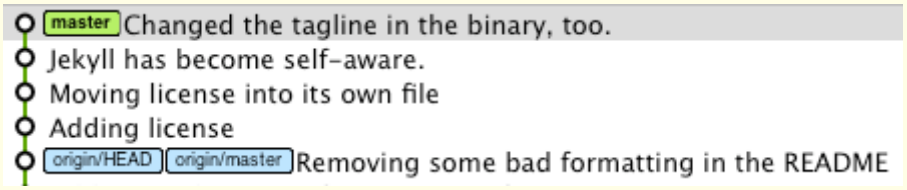
squashing commits with rebase

committed 10 Feb 2009

The rebase command has some awesome options available in its `--interactive` (or `-i`) mode, and one of the most widely used is the ability to squash commits. What this does is take smaller commits and combine them into larger ones, which could be useful if you're wrapping up the day's work or if you just want to package your changes differently. We're going to go over how you can do this easily.

A word of caution: Only do this on commits that haven't been pushed an external repository. If others have based work off of the commits that you're going to delete, plenty of conflicts can occur. Just don't rewrite your history if it's been shared with others.

So let's say you've just made a few small commits, and you want to make one larger commit out of them. Our repository's history currently looks like this:



The last 4 commits would be much happier if they were wrapped up together, so let's do just that through interactive rebasing:

```
$ git rebase -i HEAD~4

pick 01d1124 Adding license
pick 6340aaa Moving license into its own file
pick ebfd367 Jekyll has become self-aware.
pick 30e0ccb Changed the tagline in the binary, too.

# Rebase 60709da..30e0ccb onto 60709da
#
# Commands:
#  p, pick = use commit
#  e, edit = use commit, but stop for amending
#  s, squash = use commit, but meld into previous commit
#
# If you remove a line here THAT COMMIT WILL BE LOST.
# However, if you remove everything, the rebase will be aborted.
#
```

So, a few things have happened here. First of all, I told Git that I wanted to rebase using the last four commits from where the `HEAD` is with `HEAD~4`. Git has now put me into an editor with the above text in it, and a little explanation of what can be done. You have plenty of options available to you from this screen, but right now we're just going to squash everything into one commit. So, changing the first four lines of the file to this will do the trick:

```
pick 01d1124 Adding license
squash 6340aaa Moving license into its own file
squash ebfd367 Jekyll has become self-aware.
squash 30e0ccb Changed the tagline in the binary, too.
```

Basically this tells Git to combine all four commits into the the first commit in the list. Once this is done and saved, another editor pops up with the following:

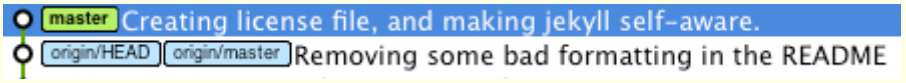
```
# This is a combination of 4 commits.
# The first commit's message is:
Adding license

# This is the 2nd commit message:

Moving license into its own file

# This is the 3rd commit message:
```

```
Jekyll has become self-aware.

# This is the 4th commit message:

Changed the tagline in the binary, too.

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# Explicit paths specified without -i nor -o; assuming --only paths...
# Not currently on any branch.
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   LICENSE
#       modified:   README.textile
#       modified:   Rakefile
#       modified:   bin/jekyll
#
```

Since we're combining so many commits, Git allows you to modify the new commit's message based on the rest of the commits involved in the process. Edit the message as you see fit, then save and quit. Once that's done, your commits have been successfully squashed!

```
Created commit 0fc4eea: Creating license file, and making jekyll self-aware.
 4 files changed, 27 insertions(+), 30 deletions(-)
  create mode 100644 LICENSE
        Successfully rebased and updated refs/heads/master.
```

And if we look at the history again…



So, this has been a relatively painless so far. If you run into conflicts during the rebase, they're usually quite easy to resolve and Git leads you through as much as possible. The basics of this is fix the conflict in question, `git add` the file, and then `git rebase --continue` will resume the process. Of course, doing a `git rebase --abort` will bring you back to your previous state if you want. If for some reason you've lost a commit in the rebase, you can use the reflog to get it back.

There's plenty of other uses for `git rebase -i` that haven't been covered yet. If you have one you'd like to share, please do so! GitCasts also has a fantastic video on this process as a whole that also covers some more complex examples of the command.