Using Git rebase on the command line

In this article

Here's a short tutorial on using `git rebase` on the command line.

Using Git rebase

In this example, we will cover all of the `git rebase` commands available, except for `exec`.

We'll start our rebase by entering `git rebase --interactive HEAD~7` on the terminal. Our favorite text editor will display the following lines:

```
pick 1fc6c95 Patch A
pick 6b2481b Patch B
pick dd1475d something I want to split
pick c619268 A fix for Patch B
pick fa39187 something to add to patch A
pick 4ca2acc i cant' typ goods
pick 7b36971 something to move before patch B
```

In this example, we're going to:

- Squash the fifth commit (`fa39187`) into the `"Patch A"` commit (`1fc6c95`), using `squash`.
- Move the last commit (`7b36971`) up before the `"Patch B"` commit (`6b2481b`), and keep it as `pick`.
- Merge the `"A fix for Patch B"` commit (`c619268`) into the `"Patch B"` commit (`6b2481b`), and disregard the commit message using `fixup`.
- Split the third commit (`dd1475d`) into two smaller commits, using `edit`.
- Fix the commit message of the misspelled commit (`4ca2acc`), using `reword`.

Phew! This sounds like a lot of work, but by taking it one step at a time, we can easily make those changes.

To start, we'll need to modify the commands in the file to look like this:

```
pick 1fc6c95 Patch A
squash fa39187 something to add to patch A
pick 7b36971 something to move before patch B
pick 6b2481b Patch B
fixup c619268 A fix for Patch B
edit dd1475d something I want to split
reword 4ca2acc i cant' typ goods
```

We've changed each line's command from `pick` to the command we're interested in.

Now, save and close the editor; this will start the interactive rebase.

Git skips the first rebase command, `pick 1fc6c95`, since it doesn't need to do anything. It goes to the next command, `squash fa39187`. Since this operation requires your input, Git opens your text editor once again. The file it opens up looks something like this:

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

Patch A

# This is the 2nd commit message:

something to add to patch A

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# Not currently on any branch.
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
# modified:   a
#
```

This file is Git's way of saying, "Hey, here's what I'm about to do with this `squash`." It lists the first commit's message (`"Patch A"`), and the second commit's message (`"something to add to patch A"`). If you're happy with these commit messages, you can save the file, and close the editor. Otherwise, you have the option of changing the commit message by simply changing the text.

When the editor is closed, the rebase continues:

```
pick 1fc6c95 Patch A
squash fa39187 something to add to patch A
pick 7b36971 something to move before patch B
pick 6b2481b Patch B
fixup c619268 A fix for Patch B
edit dd1475d something I want to split
reword 4ca2acc i cant' typ goods
```

Git processes the two `pick` commands (for `pick 7b36971` and `pick 6b2481b`). It *also* processes the `fixup` command (`fixup c619268`), since it doesn't require any interaction. `fixup` merges the changes from `c619268` into the commit before it, `6b2481b`. Both changes will have the same commit message: `"Patch B"`.

Git gets to the `edit dd1475d` operation, stops, and prints the following message to the terminal:

```
You can amend the commit now, with

        git commit --amend

Once you are satisfied with your changes, run

        git rebase --continue
```

At this point, you can edit any of the files in your project to make any additional changes. For each change you make, you'll need to perform a new commit, and you can do that by entering the `git commit --amend` command. When you're finished making all your changes, you can run `git rebase --continue`.

Git then gets to the `reword 4ca2acc` command. It opens up your text editor one more time, and presents the following information:

```
i cant' typ goods

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# Not currently on any branch.
# Changes to be committed:
#   (use "git reset HEAD^1 <file>..." to unstage)
#
# modified:   a
#
```

As before, Git is showing the commit message for you to edit. You can change the text (`"i cant' typ goods"`), save the file, and close the editor. Git will finish the rebase and return you to the terminal.

Pushing rebased code to GitHub

Since you've altered Git history, the usual `git push origin` **will not** work. You'll need to modify the command by "force-pushing" your latest changes:

```
# Don't override changes
$ git push origin main --force-with-lease

# Override changes
$ git push origin main --force
```

> Force pushing has serious implications because it changes the historical sequence of commits for the branch. Use it with caution, especially if your repository is being accessed by multiple people.

Further reading

- "Resolving merge conflicts after a Git rebase"