How do I squash two non-consecutive commits?

Asked 14 years, 2 months ago Modified 4 years, 7 months ago

Viewed 66k times

308

I'm a bit new to the whole rebasing feature within git. Let's say that I made the following commits:

```
A -> B -> C -> D
```

Afterwards, I realize that `D` contains a fix which depends on some new code added in `A`, and that these commits belong together. How do I squash `A` & `D` together and leave `B` & `C` alone?

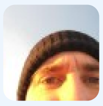
git git-rebase

Share

Improve this question

Follow

asked Oct 13, 2010 at 7:52



Nik Reiman

40.3k ● 29 ● 107 ● 161

5 Answers

Sorted by: Highest score (default)

You can run `git rebase --interactive` and reorder D before B and squash D into A.

453

Git will open an editor, and you see a file like this, ex: `git rebase --interactive HEAD~4`

```
pick aaaaaaa Commit A
pick bbbbbbb Commit B
pick ccccccc Commit C
pick ddddddd Commit D

# Rebase aaaaaaa..ddddddd onto 1234567 (4
command(s))
#
# Commands:
# p, pick = use commit
# r, reword = use commit, but edit the commit
message
# e, edit = use commit, but stop for amending
# s, squash = use commit, but meld into previous
commit
# f, fixup = like "squash", but discard this
commit's log message
# x, exec = run command (the rest of the line)
using shell
#
# These lines can be re-ordered; they are executed
from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE
LOST.
#
# However, if you remove everything, the rebase
will be aborted.
#
# Note that empty commits are commented out
```

Now you change the file that it looks like this:

```
pick aaaaaaa Commit A
squash ddddddd Commit D
pick bbbbbbb Commit B
pick ccccccc Commit C
```

And git will now meld the changes of A and D together into one commit, and put B and C afterwards. When you don't want to keep the commit message of D, instead of `squash`, you would use the `fixup` keyword. For more on `fixup`, you can consult the `git rebase` docs, or check out this question which has some good answers.

Share Improve this answer

Follow

edited Apr 28, 2020 at 6:05

rogerdpack

66.5k ● 39 ● 282 ● 401

answered Oct 13, 2010 at 7:55

Rudi

19.9k ● 3 ● 57 ● 78

---

5 Initially, I read it as "rebase D onto A, squash D into A, then rebase B onto DA". It's not clear from the answer that this can be done by reordering lines in a text editor. – Victor Sergienko May 31, 2017 at 17:56

4 If your branch is local, you will get `There is no tracking information for the current branch` error when rebasing. In this case you need to specify the number of commits you want to work with, like this: `git rebase -i HEAD~4`. See this answer. – johndodo May 16, 2018 at 10:55

10 I use interactive mode(git rebase -i) for years, I just realized it can be **reordered**. Thanks 🤟🏻 – CalvinChe Jul 2, 2019 at 9:06

1 This is to be told in bold to all who are new to git. Nobody ever told me reordering in rebase works wonders. – hardeep Mar 24, 2021 at 0:19

1 @thirdeye Of course, just put them one after each other into the list, and set all but the first of them to squash – Rudi Jul 19 at 9:43

55

Note: You should **not change commits that have been pushed** to another repo in any way *unless you know the consequences*.

`git log --oneline -4`

```
D commit_message_for_D
C commit_message_for_C
B commit_message_for_B
A commit_message_for_A
```

`git rebase --interactive`

```
pick D commit_message_for_D
pick C commit_message_for_C
pick B commit_message_for_B
pick A commit_message_for_A
```

Type `i` (Put VIM in insert mode)

Change the list to look like this (You don't have to remove or include the commit message). *Do not misspell `squash`!*:

```
pick C commit_message_for_C
pick B commit_message_for_B
pick A commit_message_for_A
squash D
```

Type `Esc` then `ZZ` (Save and exit VIM)

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

commit_message_for_D

# This is the 2nd commit message:

commit_message_for_A
```

Type `i`

Change the text to what you want the new commit message to look like. I recommend this be a description of the changes in commit `A` and `D`:

```
new_commit_message_for_A_and_D
```

Type `Esc` then `ZZ`

`git log --oneline -4`

```
E new_commit_message_for_A_and_D
C commit_message_for_C
B commit_message_for_B
```

`git show E`

```
(You should see a diff showing a combination of
changes from A and D)
```

You have now created a new commit `E`. Commits `A` and `D` are no longer in your history but are not gone. You can still recover them at this point and for a while by `git rebase --hard D` (*`git rebase --hard` will destroy any local changes!*).

Share Improve this answer

Follow

edited May 23, 2017 at 12:18

Community Bot

1 ● 1

answered Jun 21, 2013 at 15:43

Nate

13.2k ● 4 ● 63 ● 81

5

For those using SourceTree:

Make sure you haven't already pushed the commits.

1. **Repository > Interactive Rebase...**
2. Drag D (the newer commit) to be directly above A (the older commit)
3. Make sure commit D is highlighted
4. Click `Squash with previous`

Share Improve this answer Follow

edited May 6, 2016 at 2:41

answered Apr 15, 2016 at 13:34

Adam Johns
36.3k ● 26 ● 128 ● 181

1

Interactive rebase works well until you have big feature branch with 20-30 commits and/or couple of merges from master or/and fixing conflicts while you was commiting in your branch. Even with finding my commits through history and replacing `pick` with `squash` doesn't worked here. So i was looking for another way and found this article. I did my changes to work this on separate branch:

```
git checkout master
git fetch
git pull
git merge branch-name
git reset origin/master
git branch -D branch-name
git checkout -b branch-name
git add --all
#Do some commit
git push -f --set-upstream origin branch-name
```

Before this I got my pull request with about ~30 commits with 2-3 merges from master + fixing conflicts. And after this I got clear PR with one commit.

P.S. here is bash script to do this steps in automode.

Share Improve this answer Follow

edited Apr 12, 2019 at 9:40

answered Oct 10, 2018 at 6:47

Oleksiy Guzenko
41 ● 4

The first solution in that article is really nice, thanks for the link – Hoody Apr 2, 2019 at 10:20

-1

$ git checkout master

$ git log --oneline

```
D
C
B
A
```

$ git rebase --onto HEAD^^^ HEAD^

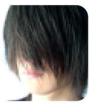
$ git log --oneline

```
D
A
```

Share Improve this answer Follow

edited Oct 13, 2010 at 15:14

answered Oct 13, 2010 at 11:17



Cotton
1,157 ● 9 ● 16

1 I think you mean `--oneline`? And it looks like you've dropped `C` and `B`, which isn't what the OP was intending. – bstpierre Oct 13, 2010 at 12:18

Didn't work for me. It moved both my HEAD and master down to A, but did not merge D into A (`git show A`) and D, C and B were lost in my ref-log. Had to `git rebase D` to get back. – Nate Jun 21, 2013 at 15:20