About Git rebase

In this article

The `git rebase` command allows you to easily change a series of commits, modifying the history of your repository. You can reorder, edit, or squash commits together.

Typically, you would use `git rebase` to:

- Edit previous commit messages
- Combine multiple commits into one
- Delete or revert commits that are no longer necessary

Warning: Because changing your commit history can make things difficult for everyone else using the repository, it's considered bad practice to rebase commits when you've already pushed to a repository. To learn how to safely rebase on GitHub.com, see "About pull request merges."

Rebasing commits against a branch

To rebase all the commits between another branch and the current branch state, you can enter the following command in your shell (either the command prompt for Windows, or the terminal for Mac and Linux):

```
git rebase --interactive OTHER-BRANCH-NAME
```

Rebasing commits against a point in time

To rebase the last few commits in your current branch, you can enter the following command in your shell:

```
git rebase --interactive HEAD~7
```

Commands available while rebasing

There are six commands available while rebasing:

`pick`

`pick` simply means that the commit is included. Rearranging the order of the `pick` commands changes the order of the commits when the rebase is underway. If you choose not to include a commit, you should delete the entire line.

`reword`
The `reword` command is similar to `pick`, but after you use it, the rebase process will pause and give you a chance to alter the commit message. Any changes made by the commit are not affected.

`edit`
If you choose to `edit` a commit, you'll be given the chance to amend the commit, meaning that you can add or change the commit entirely. You can also make more commits before you continue the rebase. This allows you to split a large commit into smaller ones, or, remove erroneous changes made in a commit.

`squash`
This command lets you combine two or more commits into a single commit. A commit is squashed into the commit above it. Git gives you the chance to write a new commit message describing both changes.

`fixup`
This is similar to `squash`, but the commit to be merged has its message discarded. The commit is simply merged into the commit above it, and the earlier commit's message is used to describe both changes.

`exec`
This lets you run arbitrary shell commands against a commit.

An example of using `git rebase`

No matter which command you use, Git will launch your default text editor and open a file that details the commits in the range you've chosen. That file looks something like this:

```
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

# Rebase 41a72e6..7b36971 onto 41a72e6
#
# 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
#
# If you remove a line here THAT COMMIT WILL BE LOST.
# However, if you remove everything, the rebase will be aborted.
#
```

Breaking this information, from top to bottom, we see that:

- Seven commits are listed, which indicates that there were seven changes between our starting point and our current branch state.
- The commits you chose to rebase are sorted in the order of the oldest changes (at the top) to the newest changes (at the bottom).
- Each line lists a command (by default, `pick`), the commit SHA, and the commit message. The entire `git rebase` procedure centers around your manipulation of

these three columns. The changes you make are *rebased* onto your repository.

- After the commits, Git tells you the range of commits we're working with (`41a72e6..7b36971`).
- Finally, Git gives some help by telling you the commands that are available to you when rebasing commits.

Further reading

- "Using Git rebase on the command line"
- The "Git Branching" chapter from the *Pro Git* book
- The "Interactive Rebasing" chapter from the *Pro Git* book
- "Squashing commits with rebase"
- "Syncing your branch in GitHub Desktop" in the GitHub Desktop documentation