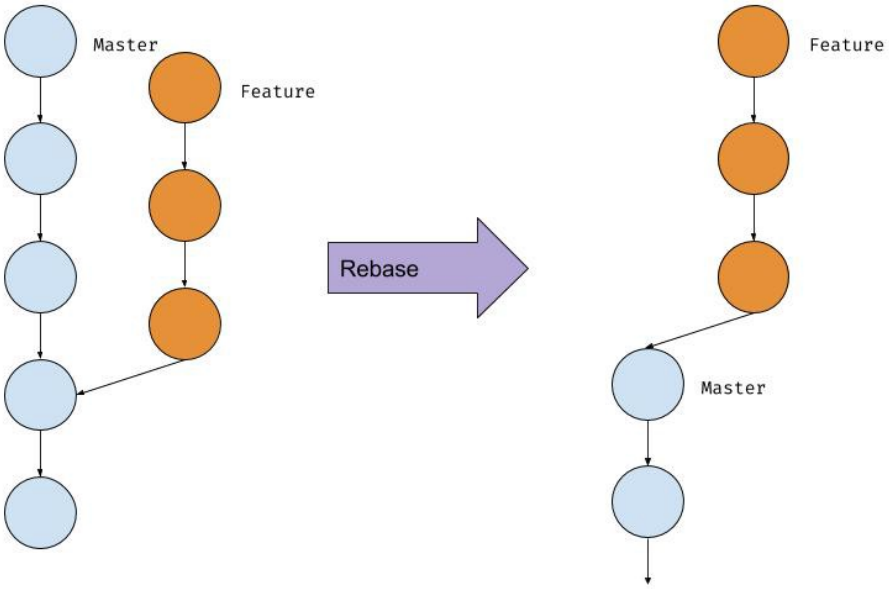
## Pre-reading

Before we dive deep into rebasing, start by reading this excellent tutorial on Atlassian. Then answer the following questions:

- What does it mean when we say merging can "pollute" our commit history?
- How does rebasing solve the problem of not needing an extra commit?
- What is the "Golden Rule" of rebasing? Why is knowing this so important?




## Rebasing Interactive Mode

Now that you have an idea of the key difference between `rebase` and `merge`, let's take a look at how rebasing in interactive mode can give us complete control over the entire commit history. You normally work in interactive mode when cleaning up a commit history before merging onto `master` or another production branch. Let's see what rebasing in interactive mode looks like. If you are using VS Code, we highly recommend you add (or already have) the following code (`export EDITOR="code -w"`) to your `.bash_profile` or `.zshrc` (if you are using `zsh`).

Let's start with a project, add a few commits and then explore the command `git rebase -i`:

```
mkdir learn_rebase
cd learn_rebase
```

```
git init

echo first > first.txt
git add .
git commit -m "adding first"

echo second > second.txt
git add .
git commit -m "adding second"

echo third > third.txt
git add .
git commit -m "adding third"

echo fourth > fourth.txt
git add .
git commit -m "adding fourth"

echo fifth > fifth.txt
git add .
git commit -m "adding fifth"

echo sixth > sixth.txt
git add .
git commit -m "adding sixth"

git log --oneline | cat | wc -l # shows us we have 6 commits

git rebase -i HEAD~5 # let's rebase from the earliest commit possible
```

We should now see something that looks like this (don't worry if your SHAs are different than this one)

```
pick 6f69a93 adding second
pick e2bb264 adding third
pick 9d46da5 adding fourth
pick dcce954 adding fifth
pick 5c040e4 adding sixth

# Rebase 77d05ef..5c040e4 onto 77d05ef (5 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
# d, drop = remove commit
#
# 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
```

As we see above, we can do some pretty awesome things with these commits. We can keep the commit, edit the commit message, edit the commit (this will pause the rebasing and we can make changes to that previous commit), squash or fixup the commit to turn a bunch of commits into one, and even remove the commit if we want. Let's see what happens if we change our rebase into this:

```
pick 6f69a93 adding second
squash e2bb264 adding third
squash 9d46da5 adding fourth
squash dcce954 adding fifth
```

```
squash 5c040e4 adding sixth

# Rebase 77d05ef..5c040e4 onto 77d05ef (5 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
# d, drop = remove commit
#
# 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
```

If we save and close the file, we will see another file open up with the following:

```
# This is a combination of 5 commits.
# The first commit's message is:
adding second

# This is the 2nd commit message:

adding third

# This is the 3rd commit message:

adding fourth
```

```
# This is the 4th commit message:

adding fifth

# This is the 5th commit message:

adding sixth

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Date:      Thu Feb 27 11:12:39 2020 -0700
#
# interactive rebase in progress; onto efb5645
# Last commands done (5 commands done):
#    squash 0e8059a adding fifth
#    squash e3b524d adding sixth
# No commands remaining.
# You are currently editing a commit while rebasing branch 'master' on 'efb5645'.
#
# Changes to be committed:
#   new file:   fifth.txt
#   new file:   fourth.txt
#   new file:   second.txt
#   new file:   sixth.txt
#   new file:   third.txt
#
```

If you change the text on the line under "# The first commit's message is:" and save and close the file, you will see only 2 commits now with a new commit message! You have squashed a bunch of commits into one larger one. This is very commonly done when working with other developers and instead of having 10-15 small commits, you can squash them down to a few larger ones for a cleaner and more accessible history.

## Rebase with GitHub

Once you have rebased, if you try to push to GitHub, your push will be rejected because the remote branch has a different commit history. In order to bypass this, you can use the `--force` (or `-f`) flag after `git push`, but be **very** careful - this will override your GitHub commit history. You **never** want to do this if other people are working on that remote branch. This is only useful if you are alone and want to push up your commits before merging into a branch that others work on. You can learn more about rebasing here.

‹ Previous

next ›