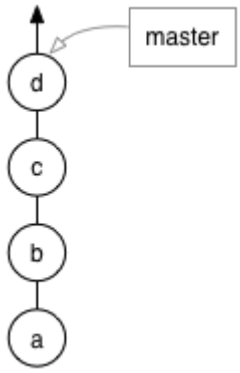
How to Rebase a Pull Request

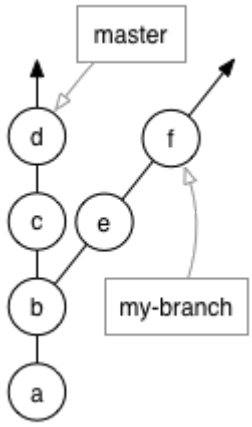
> This is a copy of article "How to Rebase a Pull Request" (https://github.com/edx/edx-platform/wiki/How-to-Rebase-a-Pull-Request) from edx-platform Wiki

When many different people are working on a project simultaneously, pull requests can go stale quickly. A "stale" pull request is one that is no longer up to date with the main line of development, and it needs to be updated before it can be merged into the project. The most common reason why pull requests go stale is due to conflicts: if two pull requests both modify similar lines in the same file, and one pull request gets merged, the unmerged pull request will now have a conflict. Sometimes, a pull request can go stale without conflicts: perhaps changes in a different file in the codebase require corresponding changes in your pull request to conform to the new architecture, or perhaps the branch was created when someone had accidentally merged failing unit tests to the master branch. Regardless of the reason, if your pull request has gone stale, you will need to rebase your branch onto the latest version of the master branch before it can be merged.

What is a rebase?

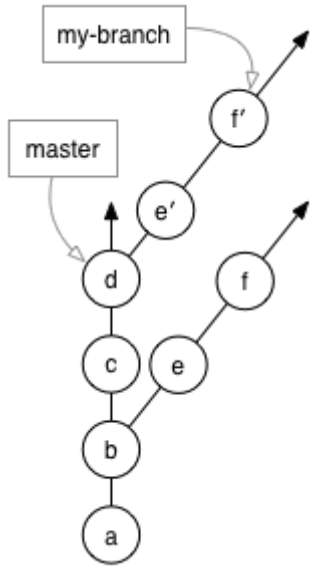


To understand this, we need to understand a bit about how Git works. A Git repository is a tree structure, where the nodes of the tree are commits. Here's an example of a very simple repository: it has four commits on the master branch, and each commit has an ID (in this case, `a`, `b`, `c`, and `d`). You'll notice that `d` is currently the latest commit (or HEAD) of the `master` branch.



Here, we have two branches: `master` and `my-branch`. You can see that `master` and `my-branch` both contain commits `a` and `b`, but then they start to diverge: `master` contains `c` and `d`, while `my-branch` contains `e` and `f`. `b` is said to be the "merge base" of `my-branch` in comparison to `master` -- or more commonly, just the "base". It makes sense: you can see that `my-branch` was based on a previous version of `master`.

So let's say that `my-branch` has gone stale, and you want to bring it up to date with the latest version of `master`. To put it another way, `my-branch` needs to contain `c` and `d`. You could do a merge, but that causes the branch to contain weird merge commits that make reviewing the pull request much more difficult. Instead, you can do a rebase.



When you rebase, Git finds the base of your branch (in this case, `b`), finds all the commits between that base and HEAD (in this case, `e` and `f`), and *re-plays* those commits on the HEAD of the branch you're rebasing onto (in this case, `master`). Git actually creates *new commits* that represent what your changes look like *on top of* `master`: in the diagram, these commits are called `eвЂІ` and `fвЂІ`. Git doesn't erase your previous commits: `e` and `f` are left untouched, and if something goes wrong with the rebase, you can go right back to the way things used to be.

Another thing to notice, however, is that Git treats branches as merely labels. The `master` branch is whatever commit the `master` label is pointing to, as well as all of that commit's parents. When you rebase a branch, Git moves the branch label to point at the newly-created commits: `my-branch` is no longer pointing at `f`, it's now pointing at `fвЂІ`. Going back to the way things used to be just consists of changing that branch label so that it points back at `f`.

Changing History

You'll notice that there is not a direct path from `f` to `fвЂІ`: from the point of view of anyone else watching `my-branch`, history has suddenly changed. In effect, `c` and `d` have been injected into `my-branch`'s history, as if they had been there the entire time. Unlike most version control systems, Git allows you to change the history of your project -- but it is very cautious about letting you do so. We'll come back to this point later.

How do I rebase?

So, now that you understand what a rebase is, the next step is learning how to do it. Let's say you have a fork of `edx-platform`, and you've created a branch, like so:

```
$ git clone https://github.com/my-username/edx-platform.git
$ cd edx-platform
$ git checkout -b my-branch
```

You've made some commits in your branch, pushed them to GitHub, and created a pull request. You're going through the code review process, responding to comments, and someone asks you to rebase your pull request. Here's what you do:

Add the official repo as a remote (first time only)

A "remote", in Git terminology, is another clone of the repository that you can share commits with. When you forked the official `edx/edx-platform` project, you created a new Git repository called `my-username/edx-platform`, but these two repositories can share commits with each other.

To add `edx` as a remote, run this command in your repository:

```
$ git remote add edx https://github.com/edx/edx-platform.git
```

You can verify that you did this successfully by running `$ git remote -v`: you should see `edx` in the list of remotes. **Note:** this step only needs to be performed once per clone.

Fetch the latest version of master

Your computer needs to download information from GitHub about the official repo, so that it knows what's on the latest version of the `master` branch. Once you've got your remote set up, this is simple. Run this command in your repository:

```
$ git fetch edx
```

Squash your changes

This step is not always necessary, but is required when your commit history is full of small, unimportant commits (such as "Fix pep8", "Add tests", or "ARUURUGHSFDFSDFSDGLJKLJ:GK"). It involves taking all the commits you've made on your branch, and squashing them all into one, larger commit. Doing this makes it easier for you to resolve conflicts while performing the rebase, and easier for us to review your pull request.

To do this, we're going to do an interactive rebase. You can also use interactive rebase to change the wording on commit messages (for example, to provide more detail), or reorder commits (use caution here: reordering commits can cause some nasty merge conflicts).

If you know the number of commits on your branch that you want to rebase, you can simply run:

```
$ git rebase -i HEAD~n
```

where `n` is the number of commits to rebase.

If you *don't* know how many commits are on your branch, you'll first need to find the commit that is base of your branch. You can do this by running:

```
$ git merge-base my-branch edx/master
```

That command will return a commit hash. Use that commit hash in constructing this next command:

```
$ git rebase -i ${HASH}
```

Note that you should *replace* `${HASH}` with the actual commit hash from the previous command. For example, if your merge base is `abc123`, you would run `$ git rebase -i abc123`. (Your hash will be a lot longer than 6 characters. Also, do NOT include a `$`!)

Once you've run a `git rebase -i` command, your text editor will open with a file that lists all the commits in your branch, and in front of each commit is the word "pick". It looks something like

this:

```
pick 1fc6c95 do something
pick 6b2481b do something else
pick dd1475d changed some things
pick c619268 fixing typos
```

For every line *except the first*, you want to replace the word "pick" with the word "squash". It should end up looking like this:

```
pick 1fc6c95 do something
squash 6b2481b do something else
squash dd1475d changed some things
squash c619268 fixing typos
```

Save and close the file, and a moment later a new file should pop up in your editor, combining all the commit messages of all the commits. Reword this commit message as you want, and then save and close that file as well. This commit message will be the commit message for the one, big commit that you are squashing all of your larger commits into. Once you've saved and closed that file, your commits have been squashed together, and you're done with this step!

Rewording commits

To reword commits, perform an interactive rebase as described in the above section. This time, instead of using the `squash` command, use the command `reword`. For example you might perform an interactive rebase and replace this:

```
pick 1fc6c95 nblargh testing new thing
pick 6b2481b Awesome New Feature: FED and Jasmine tests
```

with this:

```
reword 1fc6c95 nblargh testing new thing
pick 6b2481b Awesome New Feature: FED and Jasmine tests
```

Save and close the file, and a moment later a new file should pop up in your editor, showing you the current wording of the commit message. Reword this commit message as you want, and then save and close that file as well. This commit message will be the new commit message for your commit, thus your history can have more useful commit messages such as:

```
1fc6c95 Awesome New Feature: API Endpoints and Unit Tests
6b2481b Awesome New Feature: FED and Jasmine tests
```

Perform a rebase

To rebase your branch atop of the latest version of edx-platform, run this command in your repository:

```
$ git rebase edx/master
```

Git will start replaying your commits onto the latest version of master. You may get conflicts while doing so: if you do, Git will pause and ask you to resolve the conflicts before continuing. This is exactly like resolving conflicts with a merge: you can use `git status` to see which files are in conflict, edit the files to resolve the conflicts, and then use `git add` to indicate that the conflicts have been resolved. However, instead of running `git commit`, you instead want to run `git rebase --continue` to indicate to Git that it should continue replaying your commits. If you've squashed your commits before doing this step, you'll only have to pause to resolve conflicts once -- if you didn't squash, you may have to resolve your conflicts multiple times. If you are on Git version <2.0 and you are stuck with the message "You must edit all merge conflicts and then mark them as resolved using Git add" even though you resolved and added the file, run `git diff` and try again.

Force-push to update your pull request

As explained above, when you do a rebase, you are changing the history on your branch. As a result, if you try to do a normal `git push` after a rebase, Git will reject it because there isn't a direct path from the commit on the server to the commit on your branch. Instead, you'll need to use the `-f` or `--force` flag to tell Git that yes, you really know what you're doing. When doing force pushes, it is *highly* recommended that you set your `push.default` config setting to `simple`, which is the default in Git 2.0. To make sure that your config is correct, run:

```
$ git config --global push.default simple
```

Once it's correct, you can just run:

```
$ git push -f
```

And check your pull request. It should be updated!