Working with fixup commits

This document provides information and guidelines for working with fixup commits: - What are fixup commits - Why use fixup commits - Creating fixup commits - Squashing fixup commits

This blog post is also a good resource on the subject.

What are fixup commits

At their core, fixup commits are just regular commits with a special commit message: The first line of their commit message starts with "fixup!" (notice the space after "!") followed by the first line of the commit message of an earlier commit (it doesn't have to be the immediately preceding one).

The purpose of a fixup commit is to modify an earlier commit. I.e. it allows adding more changes in a new commit, but "marking" them as belonging to an earlier commit. `Git` provides tools to make it easy to squash fixup commits into the original commit at a later time (see below for details).

For example, let's assume you have added the following commits to your branch:

```
feat: first commit
fix: second commit
```

If you want to add more changes to the first commit, you can create a new commit with the commit message: `fixup! feat: first commit`:

```
feat: first commit
fix: second commit
fixup! feat: first commit
```

Why use fixup commits

So, when are fixup commits useful?

During the life of a Pull Request, a reviewer might request changes. The Pull Request author can make the requested changes and submit them for another review. Normally, these changes should be part of one of the original commits of the Pull Request. However, amending an existing commit with the changes makes it difficult for the reviewer to know exactly what has changed since the last time they reviewed the Pull Request.

Here is where fixup commits come in handy. By addressing review feedback in fixup commits, you make it very straight forward for the reviewer to see what are the new changes that need to be reviewed and verify that their earlier feedback has been addressed. This can save a lot of effort, especially on larger Pull Requests (where having to re-review *all* the changes is pretty wasteful).

When the time comes to merge the Pull Request into the repository, the merge script knows how to automatically squash fixup commits with the corresponding regular commits.

Creating fixup commits

As mentioned above, the only thing that differentiates a fixup commit from a regular commit is the commit message. You can create a fixup commit by specifying an appropriate commit message (i.e. `fixup! <original-commit-message-subject>`).

In addition, the `git` command-line tool provides an easy way to create a fixup commit via git commit –fixup:

```
# Create a fixup commit to fix up the last commit on the branch:
git commit --fixup HEAD ...

# Create a fixup commit to fix up commit with SHA <COMMIT_SHA>:
git commit --fixup <COMMIT_SHA> ...
```

Squashing fixup commits

As mentioned above, the merge script will automatically squash fixup commits. However, sometimes you might want to manually squash a fixup commit.

Rebasing to squash fixup commits

The easiest way to re-order and squash any commit is via rebasing interactively. You move a commit right after the one you want to squash it into in the rebase TODO list and change the corresponding action from `pick` to `fixup`.

`Git` can do all these automatically for you if you pass the `--autosquash` option to `git rebase`. See the `git` docs for more details.

Additional options

You may like to consider some optional configurations:

Configuring `git` to auto-squash by default By default, `git` will not automatically squash fixup commits when interactively rebasing. If you prefer to not have to pass the `--autosquash` option every time, you can change the default behavior by setting the `rebase.autoSquash` `git` config option to true. See the `git` docs for more details.

If you have `rebase.autoSquash` set to true, you can pass the `--no-autosquash` option to `git rebase` to override and disable this setting.