

# Committing Your Changes

Now that your staging area is set up the way you want it, you can commit your changes. Remember that anything that is still unstaged — any files you have created or modified that you haven't run `git add` on since you edited them — won't go into this commit. They will stay as modified files on your disk. In this case, let's say that the last time you ran `git status`, you saw that everything was staged, so you're ready to commit your changes. The simplest way to commit is to type `git commit`:

```
git commit
```

Doing so launches your editor of choice. (This is set by your shell's `EDITOR` environment variable — usually vim or emacs, although you can configure it with whatever you want using the `git config --global core.editor` command as you saw in [Getting Started](#)).

The editor displays the following text (this example is a Vim screen):

```
Please enter the commit message for your changes. Lines starting
with '#' will be ignored, and an empty message aborts the commit.

On branch master

Your branch is up-to-date with 'origin/master'.

Changes to be committed:
  new file:   README
  modified:   CONTRIBUTING.md

git/COMMIT_EDITMSG" 9L, 283C
```

You can see that the default commit message contains the latest output of the `git status` command commented out and one empty line on top. You can remove these comments and type your commit message, or you can leave them there to help you remember what you're committing. (For an even more explicit reminder of what you've modified, you can pass the `-v` option to `git commit`. Doing so also puts the diff of your change in the editor so you can see exactly what changes you're committing.) When you exit the editor, Git creates your commit with that commit message (with the comments and diff stripped out).

Alternatively, you can type your commit message inline with the `commit` command by specifying it after a `-m` flag, like this:

```
git commit -m "Story 182: Fix benchmarks for speed"
master 463dc4f] Story 182: Fix benchmarks for speed
2 files changed, 2 insertions(+)
create mode 100644 README
```

Now you've created your first commit! You can see that the commit has given you some output about itself: which branch you committed to (`master`), what SHA-1 checksum the commit has (`463dc4f`), how many files were changed, and statistics about lines added and removed in the commit.

Remember that the commit records the snapshot you set up in your staging area. Anything you didn't stage is still sitting there modified; you can do another commit to add it to your history. Every time you perform a commit, you're recording a snapshot of your project that you can revert to or compare to later.

---

Revision #1

Created 16 April 2019 20:10:29 by ClassCloud

Updated 16 April 2019 20:33:57 by ClassCloud