# The git commit Command

## Contents



Every time you 'add' changes (new files or existing files with some changes) and 'commit' those in your Git repository, you create a version of your project that is stored in your project history and can be accessed any time.

To commit changes with a meaning statement about changes made in a version, use `git commit` with a `-m` (m for message) flag:

```
git commit -m 'helpful statement about the change here'
```

You can see a log of your previous commits using

```
git log
```

In the log report on your terminal, you will see that each version is automatically tagged with a unique string of numbers and letters, called an SHA. You can identify, access and compare different versions by using their corresponding SHA. Here is an example of a commit in the Git log: The SHA is in the very first line, and apart from this SHA, the log also contains information on the date, time, and author of the change as well as the commit message ("minor typo fix").

```
commit 0346c937d0c451f6c622c5800a46f9e9e1c2b035
Author: Malvika Sharan <some@email.com>
Date:   Wed May 6 18:22:40 2020 +0100

    minor typo fix
```

# More on the Commit Messages

As you work on your project, you will make more and more commits. Without any other information, it can be hard to remember which version of your project is in which. Storing past versions is useless if you can not understand them, and figuring out what they contain by inspecting the code is frustrating and takes valuable time.

When you commit, you have the chance to write a commit message describing what the commit is and what it does, and you should always, *always,* ***always*** do so. A commit message gets attached to the commit, so if you look back at it (for example, via `git log`), it will show up. Creating insightful and descriptive commit messages is one of the best things you can do to get the most out of version control. It lets people (and your future self when you have long since forgotten what you were doing and why) quickly understand what updates a commit contains without having to carefully read code and waste time figuring it out. Good commit messages improve your code quality by drastically reducing wrong assumptions by people on why certain changes were made.

When you commit via `git commit` without the `-m` or `--message` option, a field appears (either within the terminal or in a text editor) where a commit message can be written. You can write a meaningful statement and save (and close if writing the message via text editor). You can set your preferred editor as the default by running a statement like this:

```
git config --global core.editor "your_preferred_editor"
```

To avoid writing this commit message in an editor, you can use the command `git commit -m "your message here"`, as discussed earlier.

## Good practice

The number one rule is: **make it meaningful**. A commit message like “Fixed a bug” leaves it entirely up to the person to understand what that means (again, this person may very well be you a few months in the future when you have forgotten what you were doing). This can end up wasting your or others time figuring out what the bug was, what changes were actually made, and how a bug was fixed. As such, a good commit message should *explain what you did, why you did it, and what is impacted by the changes*. As with comments, you should describe what the code is “doing” rather than the code itself. For example, it is not obvious what “Change N_sim to 10” actually does, but “Change number of simulations run by the program to 10” is clear.

**Summarise the changes your commit contains**. This should be written in the first line (in 50 characters maximum), then leave a blank line before you continue with the description or body of the message. The first line is the shortened version that appears as a summary when you use the command:

```
git log
```

This makes it much easier to quickly search through a large number of commits. It is also a good practice to **use the imperative present tense** in these messages. For example, instead of "I added tests for" or "Adding tests for", use "Add tests for".

Here is a good example of a commit message structure:

```
Short (50 chars. or less) summary of changes

More detailed explanatory text, if necessary. Wrap it to
about 72 characters or so. In some contexts, the first
line is treated as the subject of an email and the rest of
the text as the body. The blank line separating the
summary from the body is critical (unless you omit the body
entirely); tools like rebase can get confused if you run
the two together.

Further paragraphs come after blank lines.

  - Bullet points are okay, too

  - Typically, a hyphen or asterisk is used for the bullet,
    preceded by a single space, with blank lines in
    between, but conventions vary here
```

# Git commit: Summary

By committing your changes throughout the development of your project in meaningful units with descriptive and clear commit messages, you can create an easily understandable history. This will help you and others to understand the progress of your work. Furthermore, as the next section will demonstrate, it will also make it easy to view past versions of your history or revert changes you have made.