Writing Effective Commit Messages

> *The difference between a tolerable programmer and a great programmer is not how many programming languages they know, and it's not whether they prefer Python or Java. It's whether they can communicate their ideas. —Joel Spolsky*

Migrated from https://sites.google.com/a/nextdoor.com/wiki/engineering/commit-messages

Summary

Writing effective commit messages is an important part of good team communication. Good commit messages make it easy to know what changed from one release to the next. Good commit messages also make it easy to identify the commit to cherry-pick in order to fix a production bug, etc. Please familiarize yourself with what makes an effective commit message.

Content

The content of a commit message should clearly explain the effect of the change. A change that can't be concisely summarized in a single line might indicate that the scope of the change is too great and should be broken up into multiple commits. One-line summaries of commit messages are often sent to the whole company in the form of a release manifest, so it's good to keep them as high level as possible. Nitty-gritty engineering details can go in the detailed description below the summary.

Consistency

Just as with code, consistency in commit messages makes it easy to focus on the content rather than getting distracted by differing styles. Commit messages should start with **capital**letters, use **verbs** to describe what the commit changes and be written in the **imperative** mood (e.g., "**Add** a groups module to the news feed page.", not "~~**Adds**~~", "~~**Added**~~" or "~~**Adding**~~"). Use git-generated commit messages as a model (e.g., "**Merge** branch 'master' of review:nextdoor.com").
Consistency in the structure and line length of commit messages is also important. See below.

Format

The first line of a commit message is special. Many git-related tools (`git log --oneline`, GitHub and Gerrit to name a few) show only the first line in certain contexts. Therefore it is important that the first line contains a concise summary describing the change. It should be short (around 50 characters) and if the commit message contains additional lines, they must be separated from the first line by a blank line. A more detailed description following the summary is optional. Lines in the detailed description should be less than or equal to 72 columns.
Writing an effective but concise summary of a commit is not easy, and it takes a bit of thought and time to get right. That time and energy will pay off later when you or other engineers on the team read through your commit history.
Format taken from: http://tbaggery.com/2008/04/19/a-note-about-git-commit-messages.html

```
Capitalized, short (50 chars or less) summary

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.

Write your commit message in the present tense: "Fix bug" and not "Fixed
bug." This convention matches up with commit messages generated by
commands like git merge and git revert.

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
- Use a hanging indent

Bug: 44
Change-Id: Ia25e8535bcada77f4bbc473703348c506541fd75
```

Header lines (Bug, Change-Id, etc) are grouped together at the end of the commit message, separated from the previous section by a blank line. Like the first line, the header section is processed by tools (e.g., Gerrit search index), so having well-formed header lines is critical.

Examples

Problematic commit message examples

```
Fix bug #1024.
```

This doesn't explain what is being changed, and forces the reader to look up the bug or view the commit diff to find out. Instead, describe what the commit changes in the first line and put the bug number in the Bug: header.

```
quotes in place modals still jacked
```

Umm... what does this commit change?

```
simplify logic to handle duplicate addresses: if residence with same
address found, mark them as deleted, and the corresponding plot with
status STATUS_NO_ADDRESS -- if a corelogic residence can't find a match,
we pick the centroid of the corelogic parcel. Later we subsume all the
centroids used -- and during that process even the plots with
status-no-address could be used.
```

Good description, but not formatted properly. Some tools will truncate the message, making it more difficult to get the gist of the change.

```
Fix test.
```

Okay. Which one? How?

```
Fix typo.
```

```
copy
```

These two are also a bit vague.

```
tweak!
```

Huh?

Effective commit message examples

```
Delete unused management commands.
```

Now we're talking. This one line communicates the change clearly and concisely.

```
Incorporate city geometry into city map rendering.
```

Also concise and clear.

```
Whitelist accepted image types for profile photos.

Check the extensions of the uploaded file against a whitelist of
supported image types.

We could add client-side checks as well, but server-side checks are
good to have as a first step.

Bug: 1330
Change-Id: I1a6e00d4ff85b8a728492acf116b69c17dfa7a05
```

This one has a concise summary as well, but provides additional detail below.

More Reading

A Y-Combinator discussion on tense / grammatical mood in commit messages.
Great blog post about effective commit messages.