Well Formed Git Commit Message

Status - Published - https://www.tubemogul.com/engineering/how-to-write-well-formed-git-commit-messages/

Date Created - 5/23

Date Published - 5/26/16

Author - Ashik Uzzaman

Title

Well Formed Git Commit Messages

Tags

git, best practices, standards

Above the fold / Exec Summary

TubeMogul is growing at an incredible rate. With this rapid growth comes some lessons learned the hard way about how to work together as a, suddenly, much larger organization. TubeMogul Engineering has started using a number of tools to help us collaborate and track our progress more effectively. One of these tools is Git, the popular version control system. Read more to learn how we made Git work for us, complete with some tips and best practices.

Body

TubeMogul is growing at an incredible rate. With this rapid growth comes some lessons learned the hard way about how to work together as a, suddenly, much larger organization. TubeMogul Engineering has started using a number of tools to help us collaborate and track our progress more effectively. One of these tools is Git, the popular version control system. This post will detail an important less on, and some best practices, that we learned about how to make Git work for us.

What is a commit message and why is it important?

A commit message is a text description that developers write before submitting code changes to a version control system. Git requires any commit to have a message to accept it. Commit messages provides an easy way for a developer to gather information about a code change without reading the code. Commit messages can also act as context before reading the code. However, if the commit message is not well formed and meaningful, it doesn't serve much purpose. Well formed commit messages provide useful information for code reviewers, merge masters, quality engineers, future developers, and even executives as part of the release notes.

It's not hard (at all) to submit code with well formed commit messages, and the time and effort committed will pay off many time over when it comes to knowledge sharing with other (or when you have to revisit some old code a month later and try to figure out "what the heck was I doing here....").

What makes a commit message well formed?

There are handful of things that help make a commit message well formed. First, Git commit message are composed of 2 parts - header and body. At TubeMogul we use header line of a commit message to tell us which JIRA ticket we are committing against as well as a very short description of the change to commit. Below a sample header (or summary) line.

```
RTBSYS-375: Check how AdServer warms up its cache
```

The above commit header begins with JIRA ticket number and concludes with a descriptive statement. The JIRA ticket number is usually 10-12 characters long, 2 characters for the : and a space, and the header message here is 38 characters long. That makes it a 50 character long summary of the commit. A Git commit header should be 50 to 72 characters long in a single line. The header should resemble your JIRA ticket subject; however, a subject in JIRA can be over 200 characters, which is too long as a header for the Git commit message. On the other hand, if your header is too short, you may not be communicating effectively through it. If you use the command Git log --oneline or Git shortlog in your module, you will see the header lines from Git log history, giving a quick overview of the module in chronological order as small changes introduced over time. This is why the header should be precise but meaningful. For example “Fix bug that … ”, “Add file/feature … ” or “Increase code coverage for ...” etc.

After you wrote the header, you should to have a full blank line as a gap before starting the body of the commit message. This works as a visual separator between the two majors parts of a commit message.

Next, a detailed message about the commit serves as the body of the message. The body of a Git commit message can as long as you want, but we generally discourage people from authoring novels for commit messages. TubeMogul's commits wrap every line after 72 characters. We wrap at 72 characters as a best practice for consistency and to follow standard industry practice.

Good commit message bodies should provide answers to the following 3 questions:

1. Why is it necessary? It may fix a bug, add a feature, improve performance, reliability, or stability.
2. How does it address the issue? This should be a high-level description of what the approach was.
3. What side effect does the commit may have? This may include benchmarks, side effects, features that need focused on during testing.

Finally, provide link to any external documents, if needed, as part of the message.

```
RTBSYS-375: Check how AdServer warms up its cache

Check is we can make adpop connection better with
adserver by making adserver warm up its cache smarter.

Add TelegraphServerMetricsHandler to JMX and get rid
of slower telegraph updates due to logging anti-pattern.
Leave both file and stash appenders for now until team
members across the board gets comfortable with
logstash monitoring. Once we reach that stage, turn off
the file appender.

Look for regression impact on cache server to adpop to
adserver route including peak hours. Soak in all zones.

for further details, check out -

https://confluence.tubemogul.info/display/RTB/Telegraph
```

The example above shows a complete commit message that meets the definition of a well formed commit. Consider a commit message as a page in Git log history book, so make it reader-friendly for future generations of developers.

Best practices when creating commit messages.

1) Commit frequently during development, then clean up to make a push. If it is difficult to summarize what your commit does because it includes several logical changes or bug fixes, consider splitting into several commits. Each commit should represent a single change.

2) Avoid lazy commit messages. Funny ("Fixes all our problems") or meaningless ("asdf asdf") commit messages were common occurrences at TM. Take 2 minutes to provide good information with your commit message, you make save yourself a headache later.

3) Keep the audience in mind when writing commit messages. Your commit message will speed up the code review process, help make good release notes, give quality engineers ideas about edge cases to test, and help onboard new developers faster. Don't describe the code, describe the intent and the approach to the change.

TL;DR. When creating your commit messages, avoid making these common excuses:

- *I'm in a hurry.* Yes, but it will make referencing the change in the future much easier.
- *But it works!* It may work now, but code is always changing.
- *I am the only one working on it.* Not true, any software project is a collaborative project.
- *Do people look at Git log history?* Yes. You may not, but others will.