Commit Messages

Info

ONLY 1 JIRA issue per commit.

Think about this scenario: a developer performs a Commit for 1 new functionality and 4 bugs.

Then, here are the problems:

1. It is difficult for the reviewer to understand which code relates to which issue.
2. It takes the reviewer more time to review. You may even discourage the reviewer to look at the code submission.
3. In case 1 bug is not properly fix, then the whole commit is rejected.

Info

- Self-commits are **not** allowed.
- **NEVER** embed binary files including jar, war, tar, gz, gzip, zip in Gerrit

Reference

To submit code into Gerrit on an unfinished feature, multiple times to bring transparency and get early feedback , refer to "Submitting a Draft Feature"

Overview

The following is based on OPEN-O experiences and multiple references in the open source community.

There are tons of reasons on why it is essentials to write good commit messages. For the sake of keeping things short, here are 3 reasons:

- ONAP is **public**, so everything you do is widely visible. As you are **proud** of your work, you want to make good **impressions**.
- It **speeds up** the reviewing process.
- It helps to write good **Release Notes**.

Which Commits would you rather read?

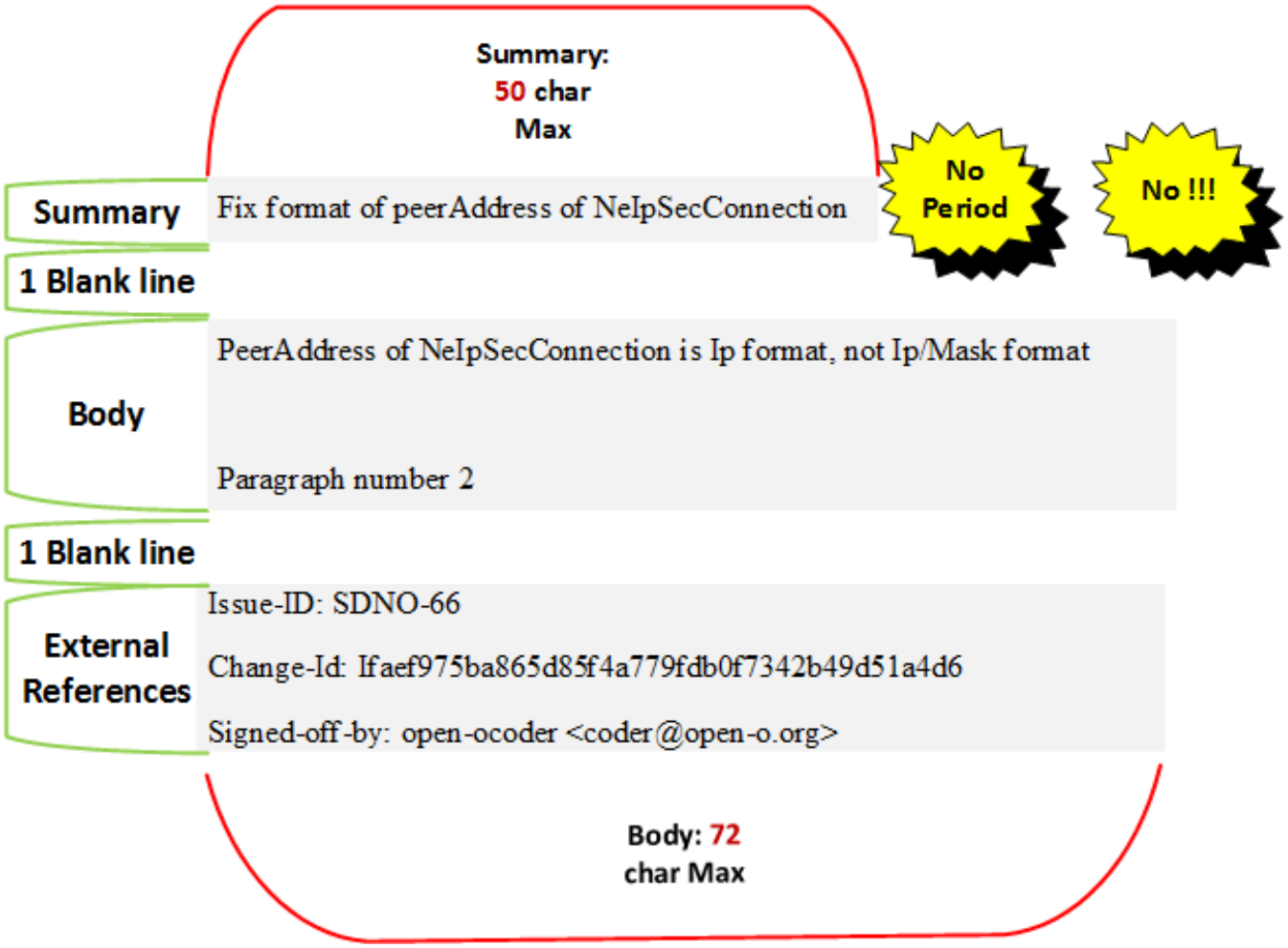
```
$ git log --oneline -5 --author cbeams --before "Fri Mar 26 2009"

e5f4b49 Re-adding ConfigurationPostProcessorTests after its brief removal
2db0f12 fixed two build-breaking issues: + reverted ClassMetadataReadingV
147709f Tweaks to package-info.java files
22b25e0 Consolidated Util and MutableAnnotationUtils classes into existin
7f96f57 polishing
```

```
$ git log --oneline -5 --author pwebb --before "Sat Aug 30 2014"

5ba3db6 Fix failing CompositePropertySourceTests
84564a0 Rework @PropertySource early parsing logic
e142fd1 Add tests for ImportSelector meta-data
887815f Update docbook dependency and generate epub
ac8326d Polish mockito usage
```

Commit Structure



Approach

To formulate good commit message focus is these 3 points:

- Explain the What, the Why and the How.
  - **What**: Do not assume the code is self-evident/self-documenting. The commit message should have a **clear statement** as to what the **original problem** is.
  - **Why**: It may fix a bug, it may add a feature, it may improve performance, reliability, stability, or just be a change for the sake of correctness. Describe the overall code structure, particularly for large changes, but more importantly **describe the intent/motivation** behind the changes.
  - **How**: Describe how the problem being fixed. Describe any limitations of the current code. For short obvious patches this part can be omitted, but it should be a high level description of what the approach was.
  - The **smaller** the amount of code being changed, the **quicker & easier** it is to review & identify potential **flaws**.

- Keep in mind than within a 3 months, you may have to review your **own** code.

Do

- Write the summary line and description of what you have done in the **imperative mode**, that is as if you were commanding someone. Start the line with a **VERB** **Fix**, **Add**, **Change** instead of Fixed, Added, Changed.
- Always leave the **second line blank**.
- Keep the summary line **shorter than 50** characters.
- Keep the body line **around 72 characters**. Make as much paragraph as you need.
- **One** commit per change.
- List **THE** issue addressed, (not a list of issues).

Don't

- Don't terminate the summary line with a **period** - it's a title and titles don't end with a period.
- Don't use **! or !!** in the message. They are useless.
- Don't provide the **issue number in the summary**. Use External Reference section to list the **ONE** Jira issue.
- Don't mix two **unrelated** functional changes. Idea is to divide to conquer. Think about the reviewer who must desiccate with issues he faces. Think about how easy it will be in case you have to revert code.
- Don't include **whitespace changes** together with code changes. Make 2 separate commits.
- Don't include **patch set-specific** comments like "**Patch set 2: rebased**"

In case of wrong commit message

There is always a possibility to change to commit message before the code is merged.

You simply have to enter the command line: git commit --amend

This method is applicable only for the latest change.

OOM Specific case

As OOM is the installer, it receives merge requests from all ONAP components. It's therefore important to understand from which component it comes and what's the specific changes it comes from.

Therefore, the commit message (on OOM) **must** follow this form:

```
[NAME_OF_COMPONENT|DOC|COMMON|GENERIC] Meaningful title (from OOM side)

at least one sentence explaining the change done in this patch, cause and consequences and possibly more of
course

Issue-ID: AS_WE_ARE_FORCED_BUT_MEANINGLESS
Change-ID: xxx
Sign-off: xxx
```

Commit message will be the last stuff that will stay with our code so it must clearly explain the changes, the "why" and the consequences. If it change OOM behavior in any way, documentation **must** be also updated.

Merge requests which are not following this pattern will not be merged.

Please read the following pages and follow the guidelines for writing commit message contained therein.

- http://bit.ly/goodcommitmessages
- http://who-t.blogspot.com/2009/12/on-commit-messages.html
- http://dep.debian.net/deps/dep3/

As you can see, most of the change is to declare which componentn is impacted by this change

References

How to Write a Git Commit Message. Very inspiring, I particularly like this sentence **What may be a hassle at first soon becomes habit, and eventually a source of pride and productivity for all involved.**
OpenStack Git commit Good Practices
Commit Message Good Practices
Writing Good Commit Messages
On Commit Message

https://gerrit.onap.org/r/Documentation/user-upload.html#push_replace (Thanks Geora)