Good commit messages

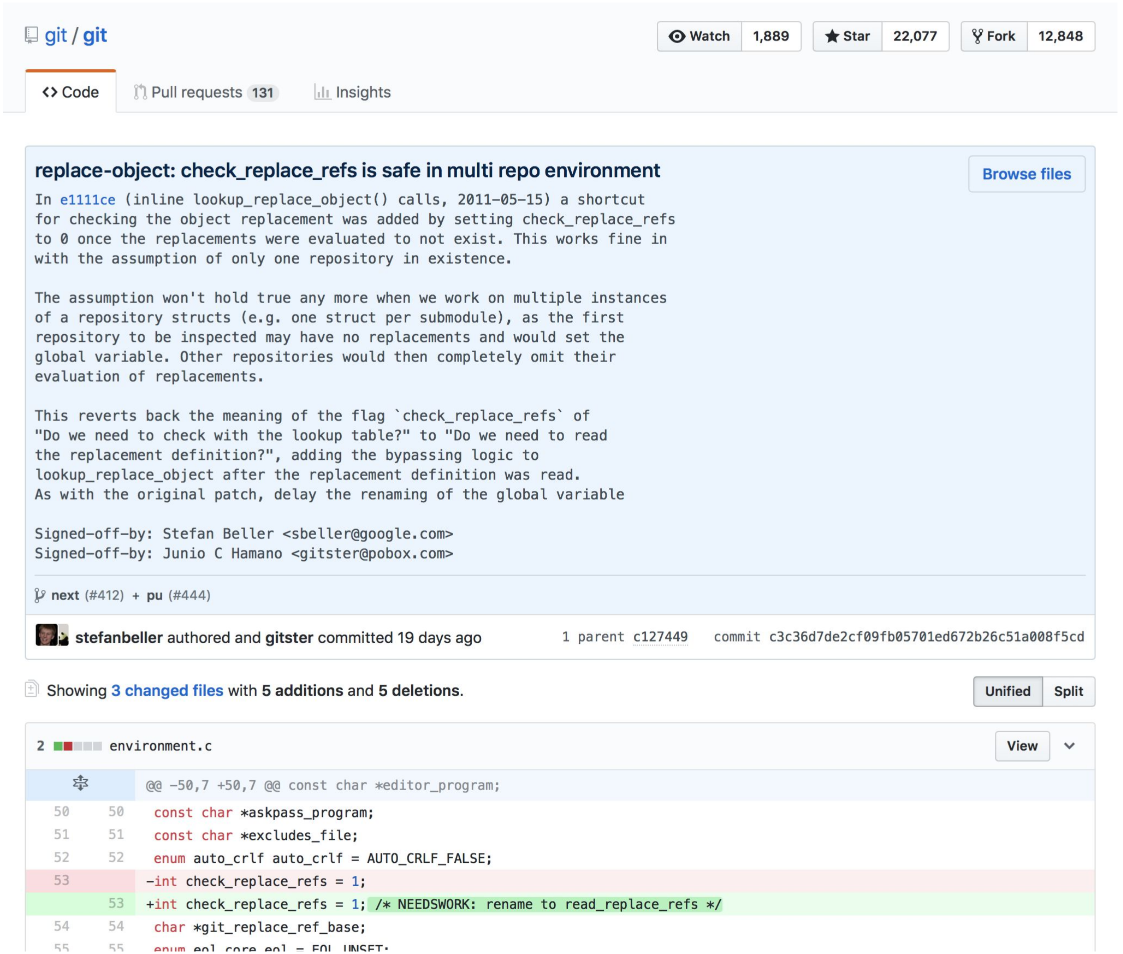
Gwyn Jones, Lead Front End Developer

Why they matter

They communicate the context of a change

*"The contributors to these repositories know that a well-crafted Git commit message is **the best way to communicate context about a change to fellow developers** (and indeed to their future selves). A diff will tell you **what** changed, but **only the commit message can properly tell you why**."*

Chris Beams. "How to Write a Git Commit Message"



git / git

Watch 1,889 Star 22,077 Fork 12,848

Code Pull requests 131 Insights

replace-object: check_replace_refs is safe in multi repo environment

Browse files

```
In e1111ce (inline lookup_replace_object() calls, 2011-05-15) a shortcut
for checking the object replacement was added by setting check_replace_refs
to 0 once the replacements were evaluated to not exist. This works fine in
with the assumption of only one repository in existence.

The assumption won't hold true any more when we work on multiple instances
of a repository structs (e.g. one struct per submodule), as the first
repository to be inspected may have no replacements and would set the
global variable. Other repositories would then completely omit their
evaluation of replacements.

This reverts back the meaning of the flag `check_replace_refs` of
"Do we need to check with the lookup table?" to "Do we need to read
the replacement definition?", adding the bypassing logic to
lookup_replace_object after the replacement definition was read.
As with the original patch, delay the renaming of the global variable

Signed-off-by: Stefan Beller <sbeller@google.com>
Signed-off-by: Junio C Hamano <gitster@pobox.com>
```

next (#412) + pu (#444)

stefanbeller authored and gitster committed 19 days ago   1 parent c127449   commit c3c36d7de2cf09fb05701ed672b26c51a008f5cd

Showing 3 changed files with 5 additions and 5 deletions.   Unified Split

2 environment.c   View

```
@@ -50,7 +50,7 @@ const char *editor_program;
 const char *askpass_program;
 const char *excludes_file;
 enum auto_crlf auto_crlf = AUTO_CRLF_FALSE;
-int check_replace_refs = 1;
+int check_replace_refs = 1; /* NEEDSWORK: rename to read_replace_refs */
 char *git_replace_ref_base;
 enum eol core_eol = EOL_UNSET;
```

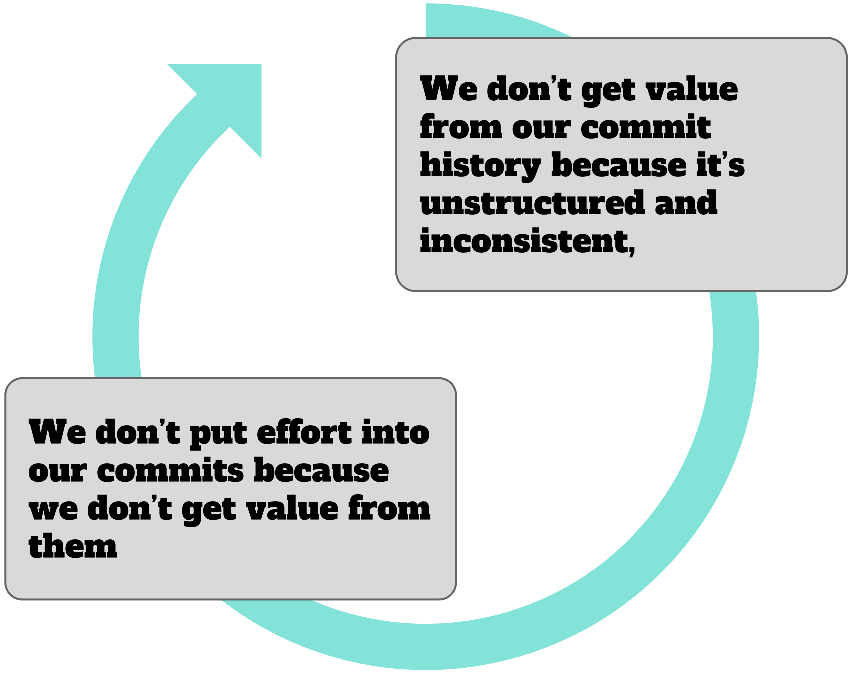
Re-establishing the context of a piece of code is wasteful. We can't avoid it completely, so our efforts should go to reducing it [as much] as possible. Commit messages can do exactly that and as a result, **a commit message shows whether a developer is a good collaborator.**

Peter Hutterer

What’s the point: a vicious cycle

“A project’s long-term success rests (among other things) on its maintainability, and a maintainer has few tools more powerful than his project’s log. It’s worth taking the time to learn how to care for one properly. What may be a hassle at first soon becomes habit, and eventually a source of pride and productivity for all involved.”

Chris Beams. “How to Write a Git Commit Message”


We don’t get value from our commit history because it’s unstructured and inconsistent,
We don’t put effort into our commits because we don’t get value from them

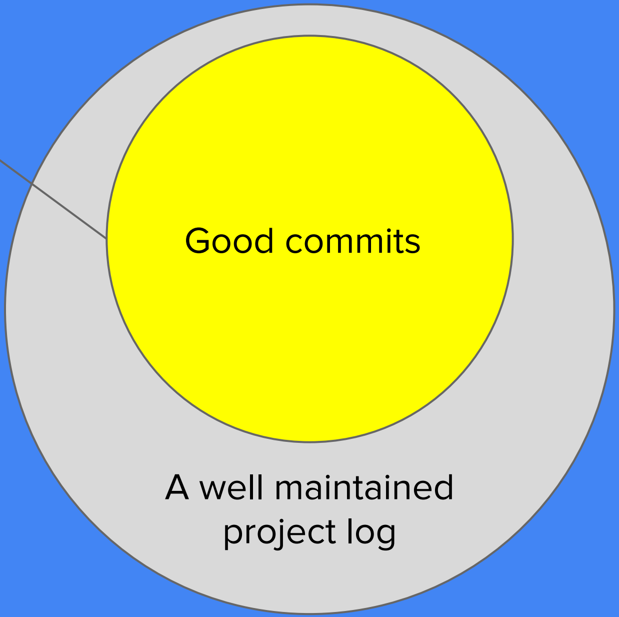
But this can easily be flipped into a virtuous cycle

Git has many really useful tools that rely upon a well maintained log

- ***Blame**** - show what revision and author last modified each line of a file
- ***Revert*** - given one or more existing commits, revert the changes that the related patches introduce, and record some new commits that record them
- ***Rebase*** - re-apply commits on top of another base commit
- ***Log*** (incl. --oneline --graph --decorate)
- ***Shortlog*** - summarises the log in a format suitable for release announcements

*like the name 'git', I believe the intention here is to be humorous. When accompanied by meaningful commit messages Git Blame becomes incredibly useful for understanding the context of a particular change.

Good commits are *part of* a well maintained project log



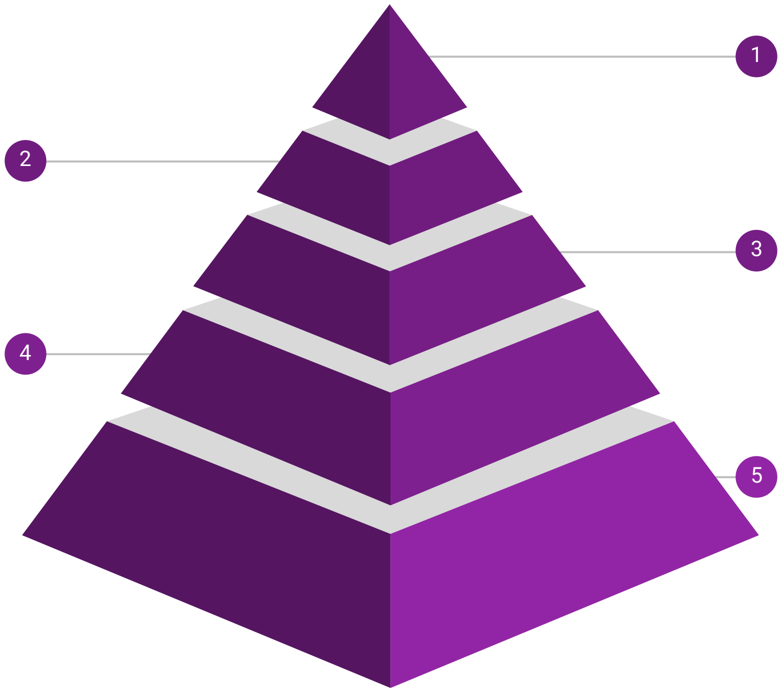
What they are

The 7 rules of a good commit message

1. Separate subject from body with a blank line
2. Limit the subject line to 50 characters
3. Capitalize the subject line
4. Do not end the subject line with a period
5. Use the imperative mood in the subject line
6. Wrap the body at 72 characters
7. **Use the body to explain *what* and *why* rather than how**

```
Summarize changes in around 50 characters or less

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 the commit 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); various tools like `log`, `shortlog`
and `rebase` can get confused if you run the two together.

Explain the problem that this commit is solving. Focus on why you
are making this change as opposed to how (the code explains that).
Are there side effects or other unintuitive consequences of this
change? Here's the place to explain them.

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

If you use an issue tracker, put references to them at the bottom,
like this:

Resolves: #123
See also: #456, #789
```

My personal hierarchy of commit 'goodness' needs



1 **Subject: capitalize and no full stop**

2 **Wrap the subject and body**

Restricting line length to 50 and 72 respectively makes for easier reading of the project log

3 **Use the imperative mood in the subject line**

This standard helps in two ways: it encourages 'atomic' commits and it helps humans parse the story of the project when it consists of commits from multiple developers

4 **Separate subject from body with a blank line**

Where the commit warrants a bit more explanation, ensuring there is a blank line between subject and the detail will help the formatting of several Git tools

5 **Explain *what* and *why* rather than how**

This is **the most important thing because it communicates the context of the change**.