tbaggery

tpope blogs here, when he blogs.

A Note About Git Commit Messages

19 Apr 2008

I want to take a moment to elaborate on what makes a well formed commit message. I think the best practices for commit message formatting is one of the little details that makes Git great. Understandably, some of the first commits to rails.git have messages of the really-long-line variety, and I want to expand on why this is a poor practice.

Here's a model Git commit message:

```
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 imperative: "Fix bug" and not "Fixed bug"
or "Fixes 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, followed by a
  single space, with blank lines in between, but conventions vary here

- Use a hanging indent
```

Let's start with a few of the reasons why wrapping your commit messages to 72 columns is a good thing.

- `git log` doesn't do any special special wrapping of the commit messages. With the default pager of `less -S`, this means your paragraphs flow far off the edge of the screen, making them difficult to read. On an 80 column terminal, if we subtract 4 columns for the indent on the left and 4 more for symmetry on the right, we're left with 72 columns.
- `git format-patch --stdout` converts a series of commits to a series of emails, using the messages for the message body. Good email netiquette dictates we wrap our plain text emails such that there's room for a few levels of nested reply indicators without overflow in an 80 column terminal. (The current rails.git workflow doesn't include email, but who knows what the future will bring.)

Vim users can meet this requirement by installing my vim-git runtime files, or by simply setting the following option in your git commit message file:

```
:set textwidth=72
```

For Textmate, you can adjust the "Wrap Column" option under the view menu, then use `^Q` to rewrap paragraphs (be sure there's a blank line afterwards to avoid mixing in the comments). Here's a shell command to add 72 to the menu so you don't have to drag to select each time:

```
$ defaults write com.macromates.textmate OakWrapColumns '( 40, 72, 78 )'
```

More important than the mechanics of formatting the body is the practice of having a subject line. As the example indicates, you should shoot for about 50 characters (though this isn't a hard maximum) and always, always follow it with a blank line. This first line should be a concise summary of the changes introduced by the commit; if there are any technical details that cannot be expressed in these strict size constraints, put them in the body instead. The subject line is used all over Git, oftentimes in truncated form if too long of a message was used. The following are just a handful of examples of where it ends up:

- `git log --pretty=oneline` shows a terse history mapping containing the commit id and the summary
- `git rebase --interactive` provides the summary for each commit in the editor it invokes
- if the config option `merge.summary` is set, the summaries from all merged commits will make their way into the merge commit message
- `git shortlog` uses summary lines in the changelog-like output it produces
- `git format-patch`, `git send-email`, and related tools use it as the subject for emails
- reflogs, a local history accessible with `git reflog` intended to help you recover from stupid mistakes, get a copy of the summary
- `gitk` has a column for the summary
- GitHub uses the summary in various places in their user interface

The subject/body distinction may seem unimportant but it's one of many subtle factors that makes Git history so much more pleasant to work with than Subversion.

Comments disabled. Email me instead.