Happy Patching

David Thompson

2015-06-25

Outline

Goals

- Improve the commit log
- Improve the quality of individual patches
- Improve the quality of pull requests
- Improve the code review workflow

Why?

- More readable history
- Easier to understand why a change was made
- Easier to `git bisect` to find breaking changes
- Easier to `git revert` those breaking changes
- Easier to review pull requests
- Faster code review loop

What's in a patch?

A patch:

- Stands alone as a single, complete, logical change
- Has a descriptive change log message
- Has no extraneous modifications (whitespace changes, fixing a typo in an unrelated file, etc.)
- Follows established coding conventions closely

Example

```
aad72327d17a1479f586af3cdb7123ffec2d9719
Author: Ricardo Wurmus <ricardo.wurmus@mdc-berlin.de>
Date:   Tue Jun 23 16:35:16 2015 +0200

    view: json: Add "location" field to JSON representation.

    * guix/web/view/json.scm (package->json): Add "location" field.

1 file changed, 4 insertions(+)
 guix/web/view/json.scm | 4 ++++

        Modified   guix/web/view/json.scm
diff --git a/guix/web/view/json.scm b/guix/web/view/json.scm
index e3f8bc1..73b78f3 100644
--- a/guix/web/view/json.scm
+++ b/guix/web/view/json.scm
@@ -24,8 +24,9 @@
   #:use-module (web uri)
   #:use-module (guix licenses)
   #:use-module (guix packages)
   #:use-module (guix profiles)
+  #:use-module (guix utils)
   #:use-module (gnu packages)
   #:use-module (guix web package)
   #:export (all-packages-json
             view-package-json
@@ -62,8 +63,11 @@
     ("name" ,(package-name package))
     ("version" ,(package-version package))
     ("synopsis" ,(package-synopsis package))
     ("description" ,(package-description package))
+    ("location" ,(last (string-split (location-file
+                                      (package-location package))
+                                     #\/)))
     ("homepage" ,(package-home-page package))
     ("license" ,(serialize-license package))
     ,@(if serialize-inputs?
           `(("inputs" ,(serialize-inputs (package-inputs package)))
```

Short Log

The first line of a commit log should:

- Be a short sentence ($\leq$ 72 characters maximum, but shoot for $\leq$ 50)
- Use imperative, passive voice ("Add awesome feature." vs. "Added awesome feature.")
- Prefix with an identifier for the general area you were working in ("tests: Fix the frob." or "gradebook: Give everyone an A.")
- Always end with a period.

Log Body

The body of a commit log should:

- Explain or justify the change
- For a bug fix, provide a ticket number or link to the ticket
- Explain what changes were made at a high level (The GNU ChangeLog standard is worth a read)
- Be word wrapped to 72 characters per line

Workflow

- Review the full diff before commiting (don't `git add` and immediately `git commit`)
- Use before commit hooks to run linters such as Rubocop
- Use your `$EDITOR`, not the `-m` flag, for writing your commit log

Pull Requests

A pull request should:

- Have a descriptive title and summary of the changes made
- Contain separate commits for logically separate changes
- Not contain any "fix up" commits ("Fix typo.", "Fix test.", "Remove commented code.")
- Be able to be thoroughly reviewed by a single person (No massive patch sets containing weeks of work by several people)

Code Review Goals

- Shared responsibility between submitter and reviewer
- Prioritize code review
- Disassociate pull requests from being strictly tied to a story/epic/task/etc.
- Make code review $\rightarrow$ QA $\rightarrow$ production phases happen faster

Programmer Workflow

- Commit as often as you'd like, but squash or otherwise rewrite your commits into logical patches before asking for code review
- Consider WIP branches ("story_XXXX", "task_XXXX", etc.) to be volatile (because they are), and anticipate that they could be rebased at any moment
- In response to feedback, squash the new "fix up" commits into the respective commit that is being fixed with an interactive rebase
- Push the new, rewritten branch with a `git push --force` (Scary! But GitHub doesn't play nicely with a safer method)

Reviewer Workflow

- Inspect patches individually as opposed to looking at the full diff in GitHub's web interface; each commit should stand alone
- Refer to coding conventions when pointing out style problems
- Follow up on changes made in response to your feedback quickly

References

- Git Patch Guidelines — http://git.kernel.org/cgit/git/git.git/tree/Documentation/SubmittingPatches?id=HEAD
- GNU Change Log Standards — https://www.gnu.org/prep/standards/html_node/Change-Logs.html
- On Code Review — http://glen.nu/ramblings/oncodereview.php
- A Note About Git Commit Messages — http://tbaggery.com/2008/04/19/a-note-about-git-commit-messages.html