The Perfect Gerrit Patch

A consumer report

FOSDEM 2021

Stephan Bergmann
Red Hat, Inc.

LibreOffice and Gerrit

- LibreOffice uses Gerrit for code review (https://gerrit.libreoffice.org)
  - ...and Jenkins for testing (https://ci.libreoffice.org)
- Can’t bypass Gerrit
  - ...but can skip Jenkins, in an emergency

Gerrit jargon

- You upload a **change** for review
- Successive versions of a change are known as **patch sets**
- Multiple changes can form a **relation chain**
- When you **submit** a change it becomes a “real” git commit

Consumers of changes

- Initially, during review of an evolving change
- After submitting:
  - Post-facto review
  - When the commit is later found to cause an issue

- Commenting on a change does not end once it gets submitted

Writing a change: Reformatting, a recurring issue

- And old, evolving code base following all kinds of different formatting styles
  - Mostly readable just fine, though
- When changing existing code, avoid random reformatting of unrelated code
  - One line actually changed, but five additional lines reformatted

```
-     if( foo< Bar >( baz ))
+     if (foo<Bar>(baz))
```

  - Distracts the reviewer
  - Complicates use of tools like `git blame` or `git log -S`

Writing a change: Reformatting, a recurring issue

- New files enforce clang-format to avoid later random reformatting
- Does *not* imply existing code should be clang-format'ed
  - clang-format does not make the code more beautiful, or more readable, or…
  - Merely an extra measure to avoid random reformatting, used where applicable
- When moving existing code to another file, retain the formatting
  - Adapt `solenv/clang-format/excludelist`
  - Helps tools like `git log --follow`

Sorting

- When adding to a list, keep the list lexicographically sorted
  - `#include` blocks, `gb_Library_add_exception_objects`, ...
  - Avoids accidental duplicates
  - Avoids merge conflicts

Links to elsewhere

- When you reference another git commit (in the commit message, in a code comment, in a Gerrit comment), give context:
  - `2fa55357223595a98c0dbc8bdb917b77a170da80 "Use OUStringChar"`
  - Helps identify the change if we ever switch VCS
  - Helps humans
- Same for other links, like bugzilla issues

Before you upload a change

- Test your change locally on at least one platform
  - `make check`
    - Avoid `--without-java`
    - Verify it covers the changed code
  - `--enable-werror`
  - Install clang-format for the git commit hook
- The loplugin warnings from Jenkins' gerrit_linux_clang_dbgutil likely suck
  - Sorry about that
  - You can locally use Clang and `--enable-compiler-plugins`, too

During review

- Use Gerrit comments to explain why you uploaded a new change set
  - …in case that is not obvious
  - Do *not* use the change’s commit message for that
- Keep rebases separate from actual changes
  - Though Gerrit tries to present a decent diff even then

Poor Jenkins

- Jenkins is a scarce resource
- Avoid excessive numbers of patch sets
  - Build locally first
- Avoid false relation chains among Gerrit changes
  - Expensive Jenkins rebuilds
  - Accidentally submitting out-of-order if it is a true relation chain after all

Before you submit a change

- Wait for Jenkins’ Verified+1
  - Resume upon spurious build failures
  - Rebase upon systematic issues with the parent
- Be careful about Jenkins’ Verified+1
  - If the Jenkins build is already a week old
    - Rebase once again
  - If the change’s parent commit is already a week old
    - `git pull` before you start to write a Gerrit change

Full Disclosure

- This was a rant
- But I love you all
- And I’ve made all the mistakes myself, too