ReviewingGuidelines

- High-level guidance
- Performing your review

Reviewing Patches in the Git Project ==========

Introduction ---------- The Git development community is a widely distributed, diverse, ever-changing group of individuals. Asynchronous communication via the Git mailing list poses unique challenges when reviewing or discussing patches. This document contains some guiding principles and helpful tools you can use to make your reviews both more efficient for yourself and more effective for other contributors.

Note that none of the recommendations here are binding or in any way a requirement of participation in the Git community. They are provided as a resource to supplement your skills as a contributor.

Principles ----------

Selecting patch(es) to review ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ If you are looking for a patch series in need of review, start by checking the latest "What's cooking in git.git" email (example). The "What's cooking" emails & replies can be found using the query `s:"What's` `cooking"` on the `lore.kernel.org` mailing list archive; alternatively, you can find the contents of the "What's cooking" email tracked in `whats-cooking.txt` on the `todo` branch of Git. Topics tagged with "Needs review" and those in the "[New Topics]" section are typically those that would benefit the most from additional review.

Patches can also be searched manually in the mailing list archive using a query like `s:"PATCH"` `-s:"Re:"`. You can browse these results for topics relevant to your expertise or interest.

If you've already contributed to Git, you may also be CC'd in another contributor's patch series. These are topics where the author feels that your attention is warranted. This may be because their patch changes something you wrote previously (making you a good judge of whether the new approach does or doesn't work), or because you have the expertise to provide an exceptionally helpful review. There is no requirement to review these patches but, in the spirit of open source collaboration, you should strongly consider doing so.

Reviewing patches ~~~~~~~~~~~~~~~~~~ While every contributor takes their own approach to reviewing patches, here are some general pieces of advice to make your reviews as clear and helpful as possible. The advice is broken into two rough categories: high-level reviewing guidance, and concrete tips for interacting with patches on the mailing list.

High-level guidance

- Remember to review the content of commit messages for correctness and clarity, in addition to the code change in the patch’s diff. The commit message of a patch should accurately and fully explain the code change being made in the diff.

- Reviewing test coverage is an important - but easy to overlook - component of reviews. A patch’s changes may be covered by existing tests, or new tests may be introduced to exercise new behavior. Checking out a patch or series locally allows you to manually mutate lines of new & existing tests to verify expected pass/fail behavior. You can use this information to verify proper coverage or to suggest additional tests the author could add.

- When providing a recommendation, be as clear as possible about whether you consider it "blocking" (the code would be broken or otherwise made worse if an issue isn’t fixed) or "non-blocking" (the patch could be made better by taking the recommendation, but acceptance of the series does not require it). Non-blocking recommendations can be particularly ambiguous when they are related to - but outside the scope of - a series ("nice-to-have"s), or when they represent only stylistic differences between the author and reviewer.

- When commenting on an issue, try to include suggestions for how the author could fix it. This not only helps the author to understand and fix the issue, it also deepens and improves your understanding of the topic.

- Reviews do not need to exclusively point out problems. Positive reviews indicate that it is not only the original author of the patches who care about the issue the patches address, and are highly encouraged.

- Do not hesitate to give positive reviews on a series from your work colleague. If your positive review is written well, it will not make you look as if you two are representing corporate interest on a series that is otherwise uninteresting to other community members and shoving it down their throat.

- Write a positive review in such a way that others can understand why you support the goal, the approach, and the implementation the patches took. Make sure to demonstrate that you did thoroughly read the series and understood problem area well enough to be able to say that the patches are written well. Feel free to "think out loud" in your review: describe how you read & understood a complex section of a patch, ask a question about something that confused you, point out something you found exceptionally well-written,

etc.

- In particular, uplifting feedback goes a long way towards encouraging contributors to participate more actively in the Git community.

Performing your review

- Provide your review comments per-patch in a plaintext "Reply-All" email to the relevant patch. Comments should be made inline, immediately below the relevant section(s).

- You may find that the limited context provided in the patch diff is sometimes insufficient for a thorough review. In such cases, you can review patches in your local tree by either applying patches with linkgit:git-am[1] or checking out the associated branch from https://github.com/gitster/git once the series is tracked there.

- Large, complicated patch diffs are sometimes unavoidable, such as when they refactor existing code. If you find such a patch difficult to parse, try reviewing the diff produced with the `--color-moved` and/or `--ignore-space-change` options.

- If a patch is long, you are encouraged to delete parts of it that are unrelated to your review from the email reply. Make sure to leave enough context for readers to understand your comments!

- If you cannot complete a full review of a series all at once, consider letting the author know (on- or off-list) if/when you plan to review the rest of the series.

Completing a review ~~~~~~~~~~~~~~~~~~~~ Once each patch of a series is reviewed, the author (and/or other contributors) may discuss the review(s). This may result in no changes being applied, or the author will send a new version of their patch(es).

After a series is rerolled in response to your or others' review, make sure to re-review the updates. If you are happy with the state of the patch series, explicitly indicate your approval (typically with a reply to the latest version's cover letter). Optionally, you can let the author know that they can add a "Reviewed-by: <you>" trailer if they resubmit the reviewed patch verbatim in a later iteration of the series.

Finally, subsequent "What's cooking" emails may explicitly ask whether a reviewed topic is ready for merging to the `next` branch (typically phrased "Will merge to 'next'?"). You can help the maintainer and author by responding with a short description of the state of your (and others', if applicable) review, including the links to the relevant thread(s).

Terminology ---------- nit: :: Denotes a small issue that should be fixed, such as a typographical error or misalignment of conditions in an `if()` statement.

aside:

optional:

non-blocking:
Indicates to the reader that the following comment should not block the acceptance of the patch or series. These are typically recommendations related to code organization & style, or musings about topics related to the patch in question, but beyond its scope.

s/<before>/<after>/
Shorthand for "you wrote <before>, but I think you meant <after>," usually for misspellings or other typographical errors. The syntax is a reference to "substitute" command commonly found in Unix tools such as `ed`, `sed`, `vim`, and `perl`.

cover letter
The "Patch 0" of a multi-patch series. This email describes the high-level intent and structure of the patch series to readers on the Git mailing list. It is also where the changelog notes and range-diff of subsequent versions are provided by the author.

On single-patch submissions, cover letter content is typically not sent as a separate email. Instead, it is inserted between the end of the patch’s commit message (after the `---`) and the beginning of the diff.

#leftoverbits
Used by either an author or a reviewer to describe features or suggested changes that are out-of-scope of a given patch or series, but are relevant to the topic for the sake of discussion.

See Also -------- MyFirstContribution

□□□ #2
□ □□□ □□□ 2026-10-31 20:05:42 UTC
□ □□□ □□□ 2026-10-31 20:05:43 UTC