Writing for code review and software

Last modified: 2026-11-13 12:24:54 (PST)

Software projects depend on short documents:

- pull requests
- issues
- commit messages
- READMEs
- code comments
- release notes

Readers of these documents often arrive without context on the change, so the rules in the other chapters apply to them too. This chapter lists the general rules that matter most here and adds a few that belong to software.

Apply the general rules

- Write in the active voice and name the actor before the action.
- Limit complex sentence structures: write one main idea per sentence and avoid piling several subordinate clauses into one.
- Cut filler and prefer plain words (Word choice).
- Scan for clichés before you post non-trivial prose. Skip the scan for code, terse status lines, and short replies, which are too short to contain a cluster of tells.
- Put lists in bullet points.

Say what changed and why

- Open a pull request description with the behavior a reader would see before and after the change.
- Explain the reason for the change in the commit message, not only the content of the diff.

- State what you checked and what you did not.
- Link the issue, the data, or the document that motivates the change.

✗ Update plotting code.

✓ Label the y-axis in original units, so the figure no longer shows log-odds.

Decide writing style here, not by imitation

The repo maintainers make each writing-style decision explicitly and record it in this book. Take a writing-style choice from this book, not from the prose around your edit. A neighboring page shows what someone wrote once, not what the maintainers decided. When this book does not say how to write something, raise the question as an issue on psw, with your recommendation, instead of copying a nearby page. Code style follows the same rule in psc.

Comment code sparingly

- Write a comment that says why the code does what it does, and let the code say what it does.
- Keep each comment as short as it can be while still giving the reason.
- Update or delete a comment when the code changes.

Check CRAN before you name an R package

Before you propose a name for an R package, check that nobody already uses it. Search these places, and report the result with the name:

- the package page on CRAN, at `https://cran.r-project.org/web/packages/<name>/index.html`, which returns 404 when the name is free;
- the CRAN archive, at `https://cran.r-project.org/src/contrib/Archive/<name>/`, because a removed package still holds its name;
- Bioconductor.

A name that CRAN has used is usually unavailable for a new package and confuses users, so pick a different name.

Write release notes for the user

- Describe each change by its effect on the user, not by the file that changed.
- Put breaking changes first, because the user must act on them.

Nest document source files by heading level

Split a long Quarto document into files that follow its headings:

- The page file holds the YAML header, the macros include, every first-level heading, and an include of the body of each first-level section.
- A section body holds that section's second-level headings, and an include of the body of each subsection.
- Continue the same way at each deeper level.
- A file holds the headings nested below its own section's heading, never that heading itself, so no file only adds one heading to one include.

Readers and reviewers then find the outline in one place, and an author edits each file as a unit. Apply the structure to a page when you edit it, and leave pages you are not editing as they are.