

Issue cheatsheet

Cheatsheet:

- How to read and categorize issues
 - Is it a bug? If so, is it regression or unexpected behavior?
 - Is it a feature request? If so, is it a “modification” or an “addition”?
- How to reproduce bugs
 - If it worked: validate the example.
 - If it failed: timebox and ask for extra steps.
 - If no reproduction provided: ask for one.
 - Can you guide the issue to a higher quality reproduction?
- How to debug issues beyond reproductions
 - Did you check to see if the bug is already fixed on the main branch?
 - Can you write an automated test to reproduce the bug?
 - Did you attempt to debug the failure to understand the root cause?
 - For regressions, did you bisect commits?
 - Slow executing and hangs can be debugged with the help of sampling the process or via reflection.
- How to give feedback on a feature request
 - Can you explain the problem being solved in your own words?
 - Did you validate the opportunity?
 - If it's a pull request, do you validate the implementation?
 - Do you have any hesitation, doubt, or dislikes you wish to express?
 - Are you asking for explicit changes?
 - Do you have alternatives you want to ask about?
 - Did you thank, connect with, or validate the poster?
 - The non-violent communication (NVC) framework is observation, emotion, need, request.

Documentation cheatsheet

Cheatsheet:

- Format
 - Is it free of grammar or spelling mistakes?
 - Are documentation syntax and formatting features properly used (Such as example code highlighting and method linking)?
- Example(s)
 - Does it contain an example (or several)?
 - Is the code executable? Can it be copied/pasted?
 - Does the example code accomplish a real task (Avoids foo/bar variables)?
- Example prerequisites
 - Is the calling context explicitly stated? (Can a new developer load and run examples?)
- Description
 - Does it explain the purpose? (Why does this code exist?)
 - Does it use common words, phrases, or use cases that people may be searching for?
- Inputs are documented
 - Are all arguments documented (Argument names, expected types, and purpose of arguments)?
 - Are environment variables or other global configurations documented?
- Outputs are documented
 - Are commonly expected exceptions mentioned?
 - Are common failure modes and caveats mentioned (For example, an external website network API is down, what behavior would happen)?

Pull request cheatsheet

Cheatsheet:

- Pull request formatting
 - Titles are 50 characters or less and use imperative (commanding) tone. i.e. “fix” instead of “fixing” and include a subject.
 - Descriptions should describe the state of the world before and after the PR. They may discuss alternative implementations.
 - Review any pull request templates.
 - Say thank you to maintainers.
- Automated checks
 - Automated checks might not trigger for your PR due to security or fraud mitigation strategies.
 - Checks didn't run.
 - Run the tests locally to ensure no problems while waiting on a review.
 - Checks failed.
 - Reproduce the failure locally if possible.
 - Investigate the failure to see if it is systemic. Check other PRs, check main.
 - Checks ran and they passed.
 - Celebrate.
- What can you do while waiting for PR comments and feedback?
 - Have you found a short-term workaround (such as making a library)?
 - Have you explored alternative implementations?
 - Can you recruit some meaningful comments from the community or co-workers?
- What do you do when someone comments on your pull request?
 - Did you get a temperature from the commenter on your change?
 - Did the commenter validate your implementation or your opportunity?
 - Did the commenter ask for explicit changes?
 - Disagree productively.

Familiarity cheatsheet

Cheatsheet:

- What are ways to get more comfortable contributing to a project?
 - Look for written contribution guides.
 - Observe maintainer actions to absorb unspoken rules.
- What kinds of contributions should you look for?
 - Small changes.
 - Provably correct changes.
 - Housekeeping tasks.
 - Changes with high public demand.
 - Not a refactoring.
- Exercises to build familiarity:
 - Review merged pull requests to understand successful bright spots.
 - Review closed (unmerged) pull requests for patterns to avoid.
 - Review “old” issues for patterns that might prevent successfully closing an issue.
 - Review “new” issues to see which types of changes get the most time and attention from maintainers.
 - Seek out positive role models of the maintainers you would like to emulate.
- COIL framework for contributions:
 - Context: Find problems through observation.
 - Opportunity: Brainstorm ways to solve those problems.
 - Implementation: Write the code, file the issue, or make the change.
 - Loop: Repeat the process until it’s done. Zoom-in or zoom-out as needed.

HOW TO OPEN SOURCE

Except where otherwise noted, this book is licensed under Attribution-NonCommercial-ShareAlike 4.0 International (CC BY-NC-SA 4.0) by Richard Schneeman.

Illustrations Copyright © 2022 Travis Stewart

Publisher	Richard Schneeman
Illustrator	Travis Stewart
Cover Designer	Travis Stewart
Editor	Ruby Ku