Team Project Abstract

2022.04.13

SWPP Practice Session

Seunghyeon Nam (with lots of derived works)

Team Project

- Develop compiler optimizations as a team!

- Each team should work on a private team repository

  - Template repository swpp202301-compiler will be shared

  - Create a separate team repo using the template

  - Each member should work on individual fork of the team repo

Collaboration Basics

- Discuss what to implement with teammates

- Implement new feature/bugfix/etc on a new branch

- Send a pull request (PR) to the upstream repository

- Get the PR reviewed by at least 2 reviewers

- Merge the PR into the upstream iff it passes the review

Discussion

- Share your plans before getting into work!

  - Helps teammates understand your work

  - Prevents duplicate works between teammates

- Use GitHub Issue to keep a record of each member's plans

- Presenting short slides can be helpful

  - Especially if you're planning on a very complex algorithm…

Pull Request

- A unit of working feature

- After each PR is merged…

  - The project must compile without any problem

  - All existing tests must pass

- PRs are 'destined' to be reviewed

  - Measures should be taken to reduce the burden of reviewing!

Line Diff

- Huge amount of changes are hard to review

- Use line diff to measure the amount of changes in code

- Generally speaking, PRs should not introduce 200+ line diff

  - This does not include comments or tests

Reducing the Diff

- Line diffs should be reduced as much as possible

- One can reduce the line diff by…

  - Splitting a large PR into multiple PRs

  - Writing concise and expressive codes

  - Following the formatting rules

Splitting the Pull Request

- Each PR is responsible for a single subject

- Common mistakes that 'bloat' the PRs are⋯

  - Implement A and format B

  - Fix irrelevant bugs A and B

  - Implement a large feature A and fix B to use A

Splitting the Pull Request

- Sometimes, a single subject may seem to require large diff

- A feature that requires complex algorithm for efficiency

  - Start with simpler implementation, then improve it with another PR

- Fixing tightly related bugs A, A' and A''

  - Fix A first, then fix A' and A'' based on merged changes

Concise & Expressive Codes

- This topic is somewhat language-specific (C++)

- Use `auto`, `decltype`, or `using statement` to hide complex types

- Look for library functions (`std::do_this()`, `llvm::doThat()`)

- Follow commonly used design patterns (visitor, builder, ⋯)

  - Template-based design patterns may be hard to use

  - But they are very powerful! (CRTP, variadic templates, type traits, ⋯)

Code Formatting

- Properly formatted code is easier to read and understand

- Also, formatting reduces diffs due to minor editing

    - Spaces, newlines, parentheses, etc

- `clang-format` can be used to apply formatting rules file-wide

    - Already installed in LLVM bin directory

Code Review

- Reviewers (teammates) will take time understanding your code

- This may look slower compared to writing the code alone

- But this process makes the code less buggy & simpler

- In the long run, you get a better program in shorter time ☺

Code Review

- Based on their understanding, reviewers may

  - Approve your code to be merged

  - Give feedbacks to suggest changes

  - Ask questions about the implementation

Giving Feedbacks

- Reviewers should look for the following criteria

  - Does this PR correctly address the issue?

  - Does this PR not introduce any UB or unnecessary copy?

  - Does this PR include proper tests to show the correctness?

  - Is there any existing library that does the same job?

  - Is there any better idiom or pattern that has the same meaning?
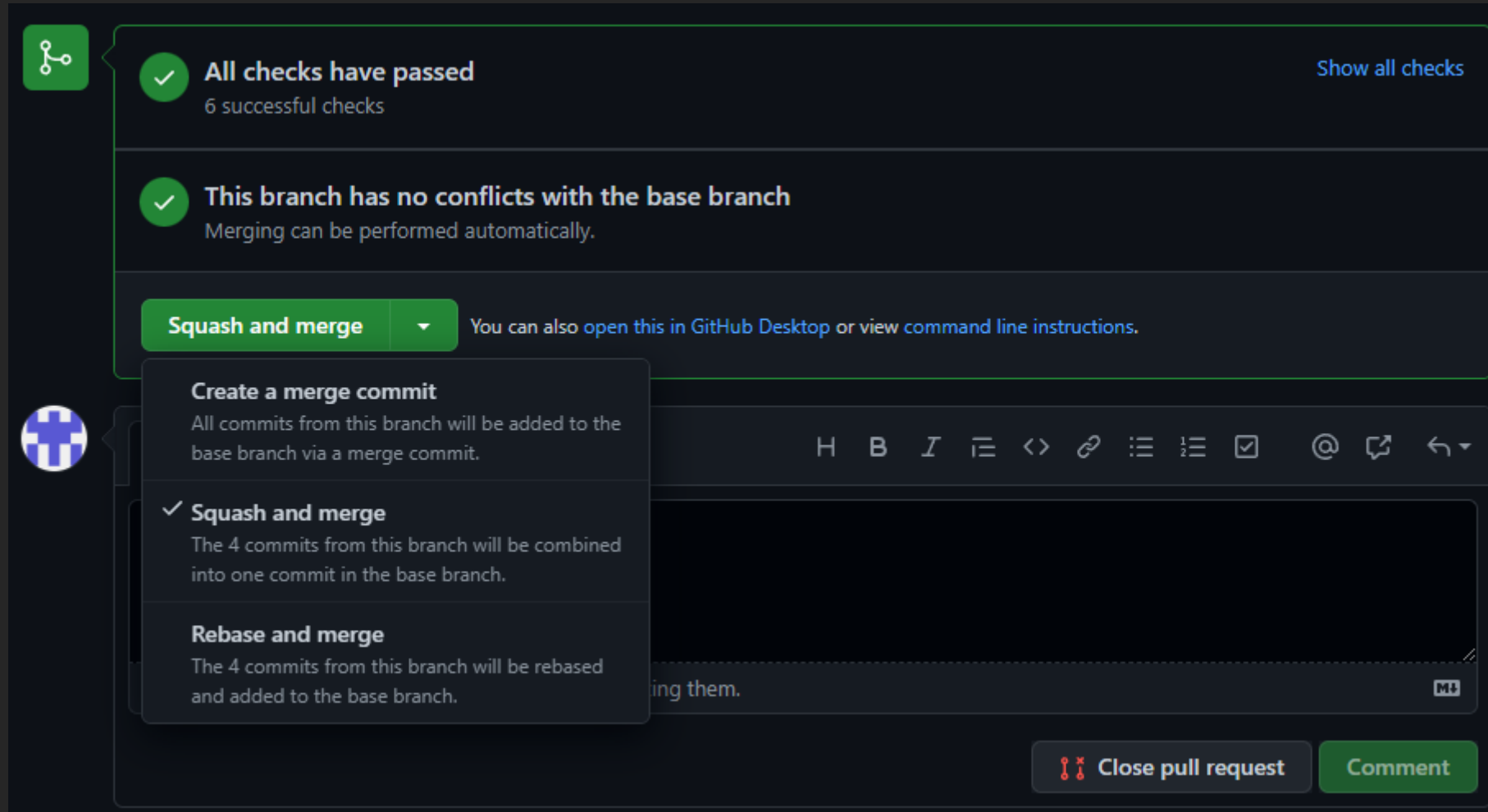
Code Review

- Feedbacks may be heartbreaking ☹

  - Remember that reviewers are not blaming you for bad code

  - Code review is an act of collaboration toward better code

- Answer the questions for you and your teammates

  - Better understanding leads to better feedbacks

  - You may 'rubber-duck' unnoticed problems in the process

Merging the Pull Request

- Use squash and merge

  - Makes the commit history concise & linear

  - Keeps the 'intermediate commits' from flooding the upstream

  - Helps reduce buggy PR into buggy commit

Merging the Pull Request