Merge, Dammit!

Steven Austin - saaustin
Sabrina Tong - stong22

Motivation

Almost every developer has struggled with git at one time or another, and one of the most frequent issues come from merge conflicts. Two or more developers will be working on the same repository at once without branching, make changes, and push before pulling each other's new code. This results in a merge conflict; the two developers cannot both push their code successfully. This results in a massive headache for the poor developer that tried to push last; they must merge their code with the head of the repository.

Some merge conflicts are significant and require careful consideration to resolve when the same portion of code has been modified. We are not trying to address this issue as there are already tools in place to do this. Some IDEs like Visual Studio code will show merge conflicted code side by side allowing programmers to make a simple choice of what version they want to keep, and we are also not trying to limit people to a single IDE. However, there is another, perhaps a more annoying type of conflict that we seek to remedy: the trivial merge conflict. Developers who have made miniscule changes that do not affect the function of their code should not be burdened with having to merge their code in git, it should happen automatically. We are trying to provide this functionality.

In our efforts to solve trivial merge conflicts for developers, we have identified three cases of trivial changes that should be auto-merged by our code. The first and most obvious case is trailing whitespace. Developers should not have to worry about an accidental space or tab creating a merge conflict, so we seek to eliminate this problem. If code is pushed that has a diff relating to trailing whitespace and would have caused a merge conflict, our code would prefer the version with no trailing spaces, and auto-fix the merge. The same would happen for pull requests, with the possibility of an extra commit to push the standardized code. The second trivial issue we see in merge conflicts is extra blank lines. They are super annoying and should not create conflicts. Our code could be configured, through either the command line or a simple UI (this is a reach goal) to either prefer to keep or prefer to discard blank lines. This will solve the blank line trivial merge issue, and work in a similar way to the trailing space issue with commits, pushing, and pulling.

The third trivial merge issue is a reach goal to address, but still trivial in the sense that it has no impact on the function of code, is the problem of internal spacing. Specifically, single spaces after brackets, parentheses, operators, if statements, try-catch blocks, etc. We want to prevent these small changes from creating merge conflicts as well, so we propose a few methods of resolving them. First, we could simply identify cases we consider to be "trivial", and then either prefer the new or old commit style when merging. Second, and more complex, users could configure the standard style they are using

for their language with regard to spacing. The auto-merger would always prefer code matching the style settings when doing an auto-merge fix.

Approach

Current merge conflict tools treat all conflicts equally rather than differentiating trivial conflicts and content conflicts. We aim to provide a tool that can differentiate the two and solve the conflict without much intervention by the user. This will by accomplished creating an interface for the user to push and pull. The application will rely heavily on the GIT API to identify merge conflicts and the lines in which they occur.

When the user pushes or pulls through our application, if a conflict exists and is determined to be trivial, it is fixed. The user is then notified of the trivial changes being made and approves the change that would align the local documents and the repository. The preferences for how conflicts are resolved can be determined through command line flags. A secondary goal would be to provide a graphical user interface to set preferences.

The aim of the application is to prevent/solve trivial merge conflicts. This requires a systemic solution to the conflict. The tool cannot be treated like a style editor as only merge conflicts will trigger an action. A user will have to go in afterwards ensure consistent style.

Challenges and Risks

One foreseeable challenge is keeping in scope. The project only has 10 weeks to come to fruition. We need to ensure that main functionality is accomplished before moving onto secondary goals. We hope to manage this by clearly identifying and differentiating primary and secondary goals. This means a lot of time will be spent in the design phase solidifying approach, requirements, and specifications. Success of the phase is highly dependent on communication. We have discussed using team communication applications like Slack to help.

A technical challenge is preventing extraneous commits. This may just be a limitation of the application depending on user preference but can be perhaps mitigated by better understanding the GIT API.