Merge Conflicts

Sam Huffer, 13 September 2018

What is a merge conflict?

Hang on, doesn't this question skip what merging is and where it occurs? Let's explain that first. And to do that, I'm going to need to explain a couple of other things.

Version Control

First, I need to explain what version control is. When you're working on a software project, you make lots of changes to code files over the course of the project. In group projects, other people are also working on the same code, making their own changes to it. Version control refers to the practice of making backups of your code after making a set of changes. This allows you to recover your code if it's accidentally deleted from your computer, or to revert to a previous backup if something goes horribly wrong with recent changes and you need to go back to the most recent working version of your code. Version control platforms like Github and BitBucket, when used with tools like Git and SourceTree, give you this functionality, affording users a sharable storage space that the group can save different versions of the project to as its development progresses. Because it's sharable, multiple people can access it simultaneously to pull changes made by other team members, making it easier to work on separate changes in parallel, which can be saved and uploaded to the server for other team members to download and further build upon.

Basic Version Control Terminology

Now that I've explained what version control is, I should briefly explain a few key terms related to version control:

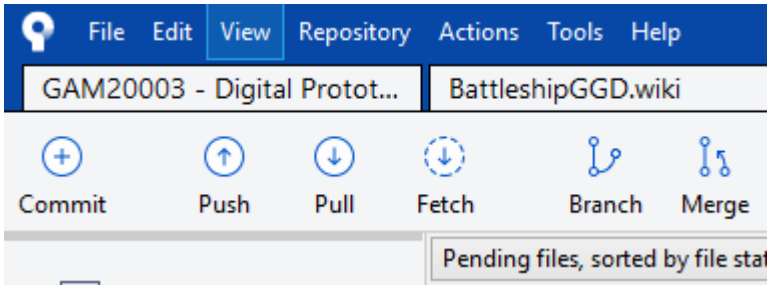

Figure 1: A panel of buttons allowing users to push, pull, fetch and commit files in SourceTree.

- Repository: the place that files are stored in by the version control system. Often shortened to just "repo", the term is usually used to refer to where files are stored on the BitBucket or GitHub server being used (the "central repository"), but it can also refer to the "local repository" that is one's computer, or more specifically the folder that the project is saved to.
- Commit: *to* commit is to save changes made to files in the local repository; *a* commit is a particular save that has been made.
- Push: to upload committed changes to the central for other people to access.
- Fetch: to check if there are any commits that others have pushed to the central repository that are ready to be downloaded.
- Pull: to download the contents of the central repository, particularly commits that have been pushed by others and that are not present on your own computer.

Pulling and Merge Conflicts

When working on a program in a group, often students will make conflicting changes to it, such as one team member modifying a file while another deletes it, or two team members modifying the same

line of code in the same file in different ways. Such conflicting modifications are typically identified when, after one student has pushed their changes to the central repository, another student attempts to pull those changes. Once the code is downloaded, the version control system being used tries to combine the two sets of changes. If there are no contradictory changes, the code is successfully merged without hassle, and the user can continue to work on their project. If there are contradictory changes, however, the version control system does not know which is correct; this is a merge conflict. The version control system will identify the contradictions, and ask the user to make changes that remove all ambiguity as to what the merged code should look like. Once the changes are made, the merge conflict is resolved and now the non-conflicting code is ready to be worked on or committed and pushed.

Merging Branches

While working on a project, a new feature might need to be added to it, but you don't want to disrupt the current version that works by adding something weird that breaks it. This is where branching comes in handy. Git and SourceTree allow users to take a commit and duplicate it, creating a new "branch". Any changes to the branch are not reflected in the original version, and changes to the original do not affect the duplicate. As such, this enables you to pursue different changes to the code in each branch, for example working on main features in the original branch while adding new functionality in the new branch. Branching also allows developers to maintain different versions of a project for different clients who need different functionality.

In the case of isolating new features in a new branch from the main working branch, once the new feature is completed and all the kinks are ironed out, the code pertaining to that feature is now ready to be transferred back into the main branch. In merging, the version control tool used takes the branch that you made your changes in and combines it with the branch you are currently in, typically the one it originated from. In doing so, it takes all the files that have been added or changed in one branch, and replicates the changes and additions in the other branch, paralleling the process by which pulled code and files are integrated with the content of the local repository. The result is (hopefully) a single, working version of the code that is stable and works as well as the original version, but also has all the additions and improvements of the new feature that was developed in the spin-off branch.

Branching and Merge Conflicts

When you create a branch, both the new branch and the original version of the project can be modified independently. Usually this is not an issue. If one change is made in one area, and other changes are made in a separate area, there won't be any issues when the different changes are reintegrated. But it can become one when two changes are made to the project that directly contradict each other, resulting in a merge conflict when one attempts to merge the opposing changes. For the branch to be successfully merged back into the original version of the code, the project will need to be modified such that this contradiction is resolved and the version control system does not perceive any ambiguity in how the final, merged version of the project should look.

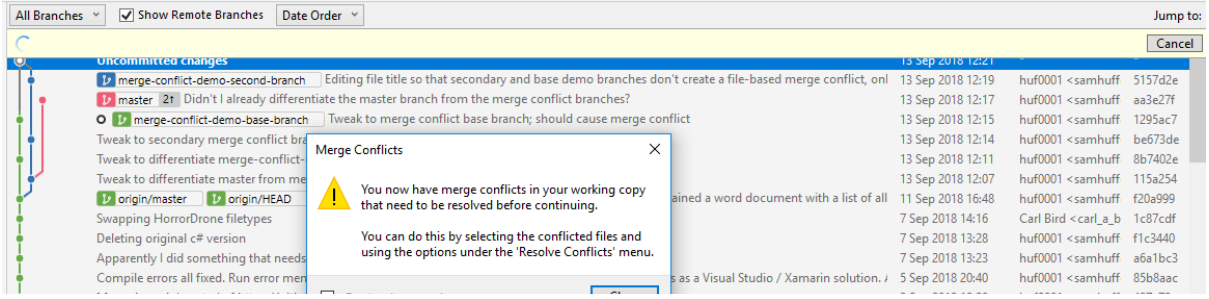

Figure 2: An error message in SourceTree notifying users of a merge conflict when they attempt to merge two branches.

Types of Merge Conflicts and Resolving Them

Regardless of the scale of the problem, or whether it occurs in pulling from the server or merging separate branches, the root cause of a merge conflict is generally the same: two different changes contradict each other, and the ambiguity-loathing version control system does not know which change to adopt when combining them. But there are different ways such contradictions can arise.

```
    // public const int FIELD_TOP = 122;
<<<<<<< HEAD
    public const int FIELD_TOP = 100;
=======
    public const int FIELD_TOP = 0;
>>>>>>> merge-conflict-demo-second-branch
    public const int FIELD_LEFT = 349;
```

Figure 3: An example of a merge conflict caused by contradicting changes to a line of code, in this case two contradicting changes to the value of FIELD_TOP.

One way merge conflict-causing contradictions can arise is through divergent changes to a line of code. Say you're working on a game in a group, and you've some tweaks to how a weapon works and have doubled its firing rate. If someone else tweaks that piece of code and halves the firing rate instead (similar to fig. 3), or even deletes that line entirely, and you try pulling their code, there is now a contradiction in changes, and therefore a merge conflict. To resolve these merge conflicts, you need to go into the files with the conflicts, locate all conflicts,

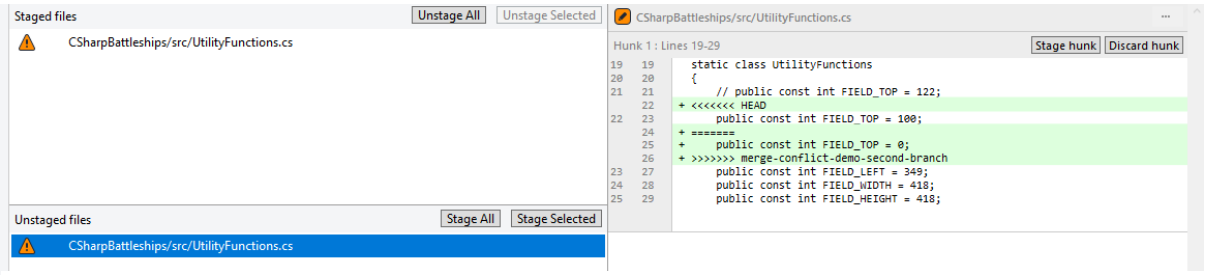

Figure 4: SourceTree highlighting which files have merge conflicts, and where in that file the conflict is.

and resolve them by choosing which of the alternate lines of code to keep, and which will not be present in the merged code.

Contradictions can also arise on the level of whole files. Say there's a file that in one branch was modified, but in another branch was deleted or renamed. When attempting to merge the two branches, the version control system will see the contradiction, and inform you of the merge conflict.

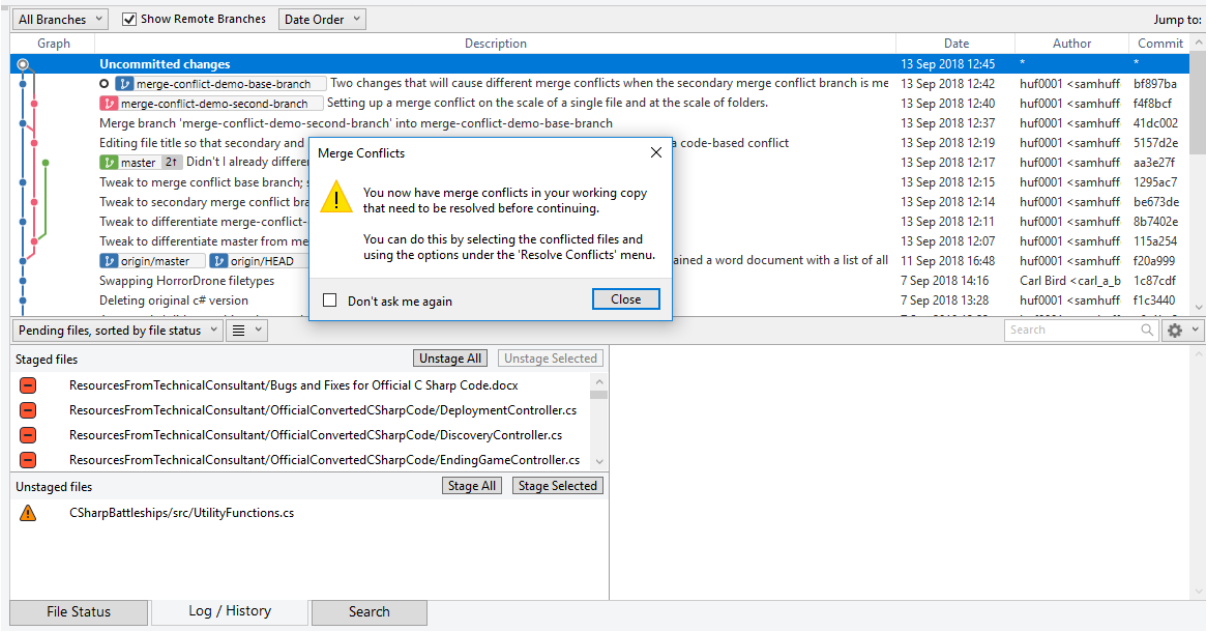

Figure 5: SourceTree complaining about merge conflicts on the level of both folders and files.

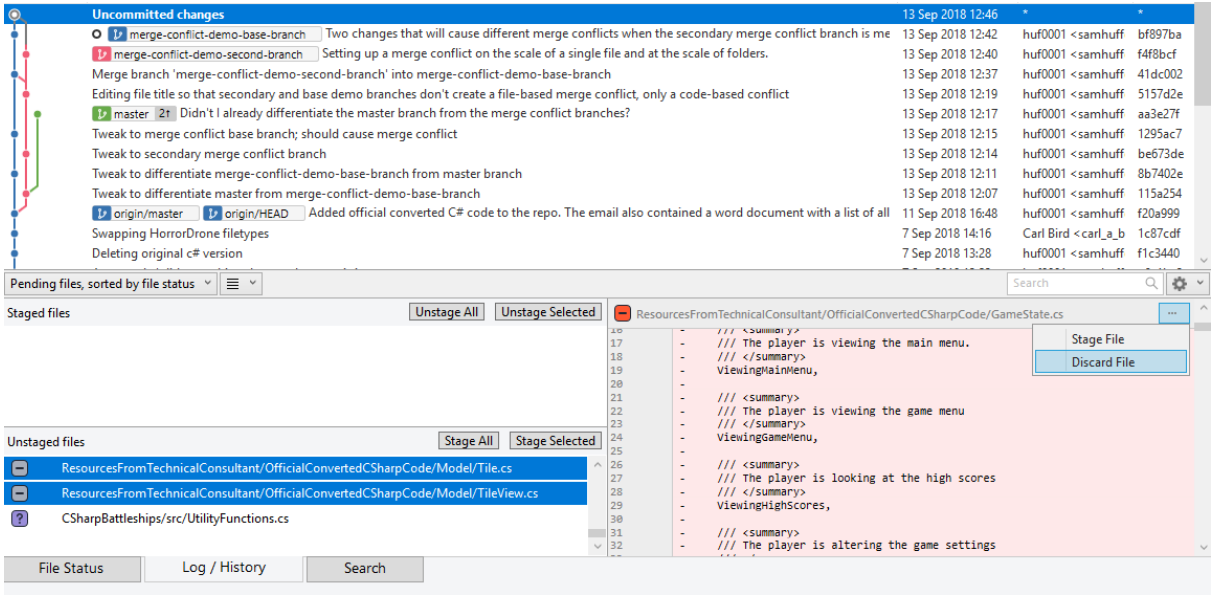

Figure 6: SourceTree when there is a merge conflict involving files or folders. Bottom left: involved files with changes waiting to be committed. Right: outline of the particular change that the user could commit. Top right: options for managing the change noted, either adding it to the list of changes to be committed ("Stage File") or dropping the changes ("Discard File").

This can also occur at the scale of folders. If you alter the contents of a folder within one branch, but rename or delete it in another, when trying to merge the two branches together, the contradictory changes will be made apparent to you by the version control system.

For this type of conflict, you will need to consider each file involved in the conflict, and choose whether you want to commit or discard a particular set of changes. For example, in the case of most files in fig. 5, the changes to be committed or discarded are the deletion of a given file from the local repository, whereas UtilityFunctions.cs is instead awaiting confirmation that modifications to it are to be committed. Each file requires you to make an individual decision on how its involvement in the merge conflict is to be handled.

Pushing and Merge Conflicts

You may have noticed that I did not discuss what might occur if someone attempted to push a commit to the central repository after someone else had pushed contradictory changes first. Well, normally you would not be able to do this at all; when you push to the central repository, if there are contradictory changes that have been pushed already, the version control system will detect this and abort the push, forcing you to pull the commits in the central repository and resolve any conflicts locally before pushing the resolved code back to the central repository (Atlassian).

You may have also noted that I said you would "normally" be unable to push to the central repository if it would create a merge conflict. That is because, when using a terminal to do all pushing, pulling, etc., you can add to the "git push" command the "--force" tag, which forces the version control system to make the central repository's contents match what is being pushed. Usually, this is only ever done when amending a commit you just made, and forcing the central repository to accept the revision before anyone else pulls its contents (Atlassian). It should never be done just to override someone else's commit, as this will cause merge conflicts that are likely more complicated and harder to resolve when they pull your code later on.

References

Atassian, *git push*, Atlassian, viewed 13 September 2018, <https://www.atlassian.com/git/tutorials/syncing/git-push >.