Git – Preventing and Handling Merge Conflicts

Devin Ellingson

Senior Software Developer, 29 Years of Professional Development Experience

Full Stack Developer specializing in Windows, .NET, and Angular as well as DevOps

Grew up in Chicago, IL, have lived in Seattle, WA, Phoenix, AZ and Las Vegas, NV

Web Site: www.devinellingson.com

Linked In: www.linkedin.com/in/devinellingson

Email: devin5885@gmail.com

Disclaimer

- There is no magic bullet for solving merge conflicts. This talk discusses some techniques I have used in my career to prevent or solve merge conflicts.

Assumptions

- That you have some experience with Git and DevOps practices.
- Development is occurring in feature branch, main (or develop) branch is branch used for deployments & builds. Assumed that main/develop branches are protected and require pull requests to update.
- There is a central git repository hosted somewhere in the cloud such as GitHub, this may be referred to as the ‘server’ in some cases.

Summary

- Avoiding Merge Conflicts - The best way to avoid merge conflicts is to prevent and minimize them in the first place

- Solving Merge Conflicts - Some techniques I have used to handle merge conflicts

- Merging Basics – How to do a merge.

Avoiding Merge Conflicts – Part 1

- Smaller merges are usually easier than large merges, therefore try to merge as often as possible (in both directions, to and from the main branch). Also, if possible don't wait till the end of the feature development cycle to merge to the main branch.
- When you create a feature branch remember to pull the latest code from the main (or develop) branch before creating the new branch. It is easy in git to create a feature branch from a main branch that is not up to date with the changes on the server.
- Avoid multiple developers working on the same code area, try to arrange development schedules so that same code is not updated by multiple developers at same time.
- Communicate with other developers about changes that you and they are making, if you both need to make the same change you may be able to synchronize your changes and thus avoid conflicts.
- Use draft pull request to make other developers aware of changes before they are ready to be merged (especially on long term projects), this can help with preventing incompatible changes appearing during a merge.

Avoiding Merge Conflicts – Part 2

- Use team-wide style standards enforced by linting or by using a style standardization tool such as Prettier (JavaScript). This eliminates annoying merge issues such as tab vs. spaces, whitespace issues etc. Also, make sure that these standards are enforced for every commit & merge.

- Review your changes before attempting to merge, remove any changes that are not needed such as accidental reformatting of code (this can occur because a change was made that turned out to not be needed). Having a merge conflict because someone split a line that they didn't change is annoying, confusing, and time-consuming.

- Avoid general code cleanup tasks on unmodified code while working on features, especially code cleanup that causes allot of small changes to multiple lines. If possible do code cleanup related tasks in areas where no feature development is occurring.

- Be careful about moving existing code, if someone else makes changes to that code the changes could easily me missed during the merge because the diffs are 'lost' due to the move.

- When deleting code make sure the code is not needed for any pending changes.

Solving Merge Conflicts – Part 1

- Remember the goal should be to get fully merged code will all features working correctly in as short a time as possible.

- Before attempting to merge your code into the main branch always merge from main branch to your feature branch and verify that your changes are compatible with and work correctly with the additional changes locally.
- Be sure to understand other developers changes before you start the merge, if necessary look at the code you are merging with and/or review the changes to that code in each commit. Trying to merge without understanding the changes on both sides could lead to breakage on both sides of the merge.
- When there is a very complex merge with a lot of changes on both sides, do a "manual" merge (even though it takes longer in the short run, could save time & problems in the long term and has a predictable end time). Also, this usually produces a better end-result and allows you to commit & test in separate commits while merging. (A manual merge is when you bypass the merge process and apply your changes manually to a new branch that is created from the latest code.

Solving Merge Conflicts – Part 2

- If you use a merge tool, configure git to use the merge tool when merging. Tools like VS Code can be used for merges w/o being configured as a merge tool in git.
- If you can, always use a GUI tool for merging, do so, command line merging is much more difficult especially for large merges.
- Don't introduce new changes during merge, this is confusing.
- You can always revert a failed merge (git merge --abort) before committing, this will return your local repository to the state before the merge started.

Merging Basics - Summary

- If possible always start with a clean working directory and index (i.e.) no uncommitted changes.
- If you need to merge with uncommitted changes stash the changes, do a pull/merge to update the branch and then merge the stash into the branch.
- The source branch is the branch (or stash) you are merging from. This changes from this branch are also referred to as ‘Incoming’.
- The target branch is the branch you are merging to. The changes from this branch are also referred to as ‘Current’ or ‘HEAD’.

Merging Basics – Merging with committed code

For this scenario we have code changes that are committed on a private (i.e.) feature branch (which may or may not be pushed to the server) and code changes in the main branch on the server These changes have conflict.

Note: To abort the merge at any time, use ‘git merge –abort’.

1. Open your merge editor (such as VS Code) to the repository.
2. Switch to the target branch using ‘git checkout’ or ‘git switch’.
3. Specify ‘git merge <Target Branch>’ to begin the merge.
4. If there are no conflicts the merge will complete and a message will be outputted indicating that.
5. If there are conflicts the files that have conflicts will not be staged, files without conflicts will be staged.
6. Use your merge editor to resolve the conflicts, file by file, conflict by conflict (or you can specify ‘git mergetool’ to automatically run your default merge tool on each conflict in each file).
7. Once all merge conflicts in the file are resolved (and the file is saved) stage the file using ‘git add’.
8. Complete the merge, using ‘git commit’.

Merging Basics – Merging with uncommitted code

For this scenario we have code changes that are not committed (i.e.) in progress and code changes in the main branch on the server. These changes conflict.

Note: To abort the merge at any time, use ‘git merge –abort’.

- Open your merge editor (such as VS Code) to the repository.
- Stash any uncommitted code (be careful about stashing untracked files they can cause merge conflicts).
- Switch to the target branch using ‘git checkout’ or ‘git switch’.
- Specify ‘git stash pop’ to merge the latest stash.
- If there are no conflicts the merge will complete and a message will be outputted indicating that, the stash will automatically be deleted.
- If there are conflicts the files that have conflicts will not be staged, files without conflicts will be staged.
- You can then use your merge editor to resolve the conflicts, file by file, conflict by conflict (or you can specify ‘git mergetool’ to automatically run your default merge tool on each conflict).
- Unstage the files that did not have merge conflicts.
- Delete the stash that was merged (this does not happen automatically if there were merge conflicts).

Appendix – Configuring default mergetool to VSCode

- Follow these steps from a PowerShell window to use VSCode as your default mergetool:
  - git config --global merge.tool vscode
  - git config --global mergetool.vscode.cmd "code --wait --merge $REMOTE $LOCAL $BASE $MERGED"

Slides/Demo Instructions

- https://github.com/devin5885/Presentations/tree/main/VegasProgrammers/02-17-2024-GitHandlingMergeConflicts