**10 – Resolving Merge Conflicts**

**Activities**

COMP190 – Tools and Techniques for Software Development

Dickinson College

**Name:**

Top of FormThis week’s topic focused on merge conflicts, why they arise and how they can be resolved. We saw that maintainers are regularly merging changes from contributor’s pull requests into the upstream main branch. If those changes are merged after you last synchronized and created your feature branch, it is possible for a merge conflict to arise. If your changes and those merged into main affect different parts of the project, then the maintainers will still be able to merge your changes automatically. However, if the changes merged into main and the changes in your feature branch alter the same code or documentation, then a merge conflict will occur. Usually, you as the contributor and the expert on your work will be asked to resolve the merge conflict so that the maintainers can merge your pull request automatically. We saw that this can be done using a graphical merge tool. In completing A09, you will have created a pull request that has a merge conflict. This set of activities takes you through the process of resolving that merge conflict.

**Merge conflict concepts:**

1. Consider the following merge. The main branch has been updated to correct the operation of the program. The feature branch has been updated to use better variable names. The center pane shows the best common ancestor of the feature and main branches.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
|  | total=0  count=0  read n  while count > n:  read m  total=total+m  count--  average=count/total |  | tot=0  count=0  read n  while count > n:  read m  tot=tot+m  count--  ave=count/tot |  | tot=0  count=0  read n  while count < n:  read m  tot=tot+m  count++  ave=tot/count |  |
|  | **Feature Branch** |  | **Common Ancestor** |  | **main Branch** |  |
|  |  |  |  |  |  |  |

a. Use the highlight tool to mark the lines in the feature branch and/or the main branch above as indicated below. Use the example in the slides as a guide for the highlighting.

i. Highlight lines that cause a non-conflicting change in blue.

ii. Highlight lines that cause a conflicting changes in red.

b. Would the above merge be able to be completed automatically? Briefly explain your answer.

c. Give a version of the above program that merges the feature branch and the main branch such that the result both operates correctly and uses improved variable names.

2. It is possible that a merge that contains only non-conflicting changes, and thus could be merged automatically, might still break the operation of a program.

a. Using the common ancestor below, fill in the feature branch and the main branch with some changes in each, such that there are no merge conflicts, but that the resulting program does not work correctly.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
|  |  |  | // swap x and y  tmp = x  x = y  y = tmp |  |  |  |
|  | **Feature Branch** |  | **Common Ancestor** |  | **main Branch** |  |
|  |  |  |  |  |  |  |

b. Highlight the lines in your example above as was indicated in #1 b.

**Synch with the Upstream main:**

In class we saw that some changes had been merged into the upstream main after you had last synched. The changes that were merged were designed to conflict with the changes required for each of the 190-round2 issues. Thus, the pull request that you made at the end of A09 will now contain conflicts that prevent it from being merged automatically. This situation is shown in Figure 1.

Figure 1 - Conflicting Upstream Changes

The activites in this section will have you confirm that you are out of synch with the upstream and that your pull request cannot be merged automatically. It will then have you synch the main branch in your local and origin repos with the upstream, so that you can resolve the merge conflict.

3. Look at the main page for your origin repo on GitHub. Be sure that the main branch is active. How can you tell from this page that there have been changes to the upstream main branch that you have not yet synched?

4. Now find your pull request in the upstream repo: <https://github.com/dickinson-comp190/FarmData2-190F21>. Can your pull request be merged automatically? Briefly explain why or why not.

5. Synch the main branch of your local and origin repos with the upstream. Don’t forget to make sure your main branch is active. What commands did you use? You can tell if this worked by checking your main GitHub page. It should now tell you that “This branch is up to date with dickinson-comp190:main”.

**Merge main Branch into Feature Branch:**



Figure 2 - Merge main into feature branch

Your local main branch now contains the changes that were made to the upstream main branch. The next step is to merge the changes from the main branch into your feature branch, as illustrated in Figure 2.

6. By default, git will not include the information about the common ancestor when it adds merge conflict information to files. It only includes the information about the active branch and the branch that is being merged (e.g. main). To configure git to include the common ancestor, as we saw in class, use the following command:

git config --global merge.conflictstyle diff3

Give the output of a git config -l (ell not one) command after issuing the above command.

7. In order to merge main into your feature branch, your feature branch must be the active branch. Make your feature branch the active branch. What command did you use? You can use git status to confirm that your command worked.

8. The git merge <branch name> command will attempt to merge the specified branch into the active branch.

a. Give a command that will merge the main branch into your feature branch.

b. What output did your command from part a generate?

9. The output from #8.b should indicate that the automatic merge failed due to a conflict.

a. In which file is the conflict located?

b. Use the cat command to display the file containing the conflict. Find the part of the file that indicates the conflict and copy and paste it here. Include the chevrons at the top and bottom of the conflict.

10. Answer the following questions about the conflicting content based on your answer to #9.b.

a. What was the content before any changes were made?

b. What content did the maintainers merge into the main branch?

c. What was the change that you made?

11. Sometimes when you try to merge, and it fails due to a conflict you may want to undo the attempted merge. Use the git merge --abort command to undo the merge. Use the cat command to display the file that contains the conflict again. How did it change?

**Install a Graphical Merge Tool:**

Before you undid the merge (in #11) you could have just used a text editor to resolve this simple conflict. But it is useful to gain practice using a graphical merge tool for when conflicts get more complicated. So, in this section you will install the Meld merge tool (<https://meldmerge.org/>) and configure git to use it when it is necessary to manually resolve merge conflicts. Then in the next section you’ll use Meld to resolve the conflict.

12. Use apt to install the Meld merge tool. What command did you use? Hint: Type the command meld for some help.

13. Use the commands below to configure git so that it will use Meld as the merge tool for manual resolution of conflicts:

git config --global merge.tool meld  
git config --global mergetool.keepBackup false

Give the output of a git config -l command after issuing the above commands.

**Resolving a Merge Conflict:**



Figure 3 - Resolving a Merge Conflict

This section will walk you through the full process of resolving the merge conflict using Meld. This flow of this process is illustrated in Figure 3. You will switch to your feature branch, perform the merge, use Meld to resolve the resulting conflict, and then stage and commit the merged content. In the next section, you’ll push that change to your origin to update your pull request.

14. Ensure that you are on your feature branch. Then issue the command to merge the main branch into your feature branch (see #8). The merge should once again fail because of the conflict. Use the cat command to confirm that the merge conflict information now appears in the conflicted file again. No answer is necessary here.

15. When a merge fails, git places the merge conflict information into the conflicted files as we have seen. The git mergetool command will launch the merge tool (e.g. Meld) that we configured so that the conflict can be resolved. Issue the git mergetool command. Paste a screen capture of the Meld window here before doing anything to its content.

15. When using Meld you resolve the conflict by modifying the center pane so that it appears as desired. You can click the arrows to move content between the panes and you may also directly edit the center pane.

Use Meld to resolve the conflict so that the merged result will contain:

* your changes where there is a conflict.
* all of the other non-conflicting changes from the main branch.

Save your changes and include a screen shot of the Meld window with the center pane showing the resolved conflict.

16. Close the Meld tool and issue a git status command. What output is generated?

17. From #16 you can see that you now have uncommitted changes. Those changes are all of the changes you made to the common ancestor to perform the merge. Stage and commit the changes you made in resolving the conflict. Give the commands that you used. Be sure to use a meaningful commit message.

**Update your Pull Request:**

At this point you have resolved the merge conflict on the feature branch in your local repository. What is left is to push that branch to your origin. When you do so, GitHub will automatically update your pull request to the upstream for that branch.

18. Push your modified feature branch to your origin. What command did you use?

19. Now find your pull request again in the upstream repo: <https://github.com/dickinson-comp190/FarmData2-190F21>. Can it be merged automatically now? Briefly explain why or why not.

**Reflection and Summary:**

20. Complete the table below by filling in the right-hand column with the commands that accomplish the task listed on the left. Use the <…> notation appropriately to indicate parameters that need to customized for each use. Note that the tasks listed are in approximately the same order as they appear in this activity.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | **Task to Complete** | **Git Commands** |  |
|  | Merge changes from one branch into another branch |  |  |
|  | Undo a merge that failed due to conflicts |  |  |
|  | Launch the configured graphical merge tool |  |  |
|  |  |  |  |

**More Practice:**

21. This section is optional but will provide you with more practice with git, GitHub and with resolving merge conflicts if you would like it.

a. Use the following command to get the mergePractice feature branch from the upstream to your local repo:

git fetch upstream mergePractice

The mergePractice branch was created so that it contains some conflicts with the upstream main branch. Pulling it to your local repo simulates the process of creating a new branch and making changes to the file – like you did earlier but with less work.

b. Make the mergePractice branch active and try to merge main into it. What commands did you use?

c. The mergePractice branch contains a few conflicts so the merge you attempted in part b should fail. Resolve the conflicts using the changes in the mergePractice branch. Also accept all of the non-conflicting changes from the main branch.

d. Push the mergePractice branch to your origin and create a pull request for it. What is the title and number of your pull request?

**Optional:** To help us improve and scope these activities for future semesters please consider providing the following feedback.

a. Approximately how much time did you spend on this activity outside of class time?

b. Please comment on any particular challenges you faced in completing this activity.

**Acknowledgements:**

Some materials, questions and resources have been adapted from activities posted on foss2serve.org:

* <http://foss2serve.org/index.php/Git:_GitHub_Issues_and_Pull_Requests>
* <http://foss2serve.org/index.php/Git:_GitHub_Workflow_Activity>
* <http://foss2serve.org/index.php/Work_Locally_with_Git_from_the_Command_Line_(Activity)>