**4 –Merge Conflicts**

**Activities**

**Name:**

Top of FormThis 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 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 by manually editing the raw merge conflict information or by using a graphical merge tool. In completing the most recent homework, you will have created a pull request that, after other work was merged into the upstream main branch, has resulted in a merge conflict. This set of activities takes you through the process of resolving that merge conflict.

**Merge conflict concepts:**

1. Consider the merge shown below. As shown in the right-hand pane, the maintainers have merged commits into the main branch that fix several bugs that existed in the program. The left-hand pane shows a contributor’s feature branch with has been changed to use more descriptive variable names but that has not fixed the bugs. 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** |  | **Best Common Ancestor** |  | **main Branch** |  |
|  |  |  |  |  |  |  |

a. Study the program in the main branch and then describe in a sentence the computation that the program above is trying to perform. Note: The feature Branch and the Best Common Ancestor both contain bugs.

b. 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 all lines containing non-conflicting changes in blue.

ii. Highlight all lines that contain conflicting changes in red.

Nothing is required here. Just be sure to highlight the code above as indicated.

c. Would the feature branch above be able to be merged automatically by the project maintainers? Briefly explain your answer.

d. Give a Merged Version of the above program that combines the feature branch and the main branch such that the result both performs the desired computation and uses the more descriptive variable names.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
|  | 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=tot/count |  |
|  | **Feature Branch** |  | **Merged Result** |  | **main Branch** |  |
|  |  |  |  |  |  |  |

2. Now consider the following merge:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
|  | r = 15  pi = 3.1415927  rsq = r\*\*2  a = pi\*rsq  print a |  | r = 15  pi = 3.14  rsq = r\*2  a = pi\*rsq  print a |  | r = 15  pi = 3.14  rsq = r\*2  area = pi\*(rsq/2)\*\*2  print area |  |
|  | **Feature Branch** |  | **Best Common Ancestor** |  | **main Branch** |  |
|  |  |  |  |  |  |  |

a. Study the program in the Feature Branch and describe in a sentence the computation that the program above is trying to perform. Note: The \*\* indicates exponentiation (e.g. x\*\*2 is x squared). Note: The Common Ancestor contains a bug that is fixed in different ways by the main branch and the feature branch.

b. Highlight the above merge in the same way you did in question #1b.

Nothing is required here. Just be sure to highlight the code above as indicated.

c. Briefly explain why this merge could be performed automatically.

d. Fill in the Merge Result column below to show the result of an automatic merge of the feature and main branches.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
|  | r = 15  pi = 3.1415927  rsq = r\*\*2  a = pi\*rsq  print a |  |  |  | r = 15  pi = 3.14  rsq = r\*2  area = pi\*(rsq/2)\*\*2  print area |  |
|  | **Feature Branch** |  | **Merged Result** |  | **main Branch** |  |
|  |  |  |  |  |  |  |

e. Look carefully at your result in part d. Will the resulting program perform the computation correctly? Briefly explain why or why not.

f. When a merge can be completed automatically, does it guarantee that the Merged Result will be correct?

**Getting the KitClient Started Again:**

Like the last several activities, you will be working within the *KitClient* for this activity.

3. Revisit one of the prior activity sheets to find the commands that you used to start the KitClient.

Nothing is required here, but you must have the KitClient running to work with your local copy of the FarmData2 repository.

**Synch with the Upstream main:**



Figure - Conflicting Upstream Changes

In class we saw that the maintainers had merged some changes into the upstream main after you had created your feature branch. The changes that were merged were specifically designed to conflict with the changes required for each of the Round2 issues. Thus, the pull request that you made at the end of the previous activity will now contain conflicts that prevent it from being merged automatically. This situation is shown in Figure 1, where the maintainers have merged the dark blue commit into the main branch.

The activities in the remainder of 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.

4. Which of the commits (i.e. which colors) in Figure 1 might contain merge conflicts after the dark blue commit was merged?

5. Visit the main page for your origin repo on GitHub and make sure that the main branch is active. You should be able to tell from this page that there have been changes to the upstream main branch that you have not yet synched (i.e. you are behind).

Give a screen shot of the part of your origin repo page on GitHub that shows that your main is out of synch and how many commits behind the upstream you are.

6. Now find your pull request in the upstream repo on GitHub. You should see that your pull request cannot be merged automatically.

If GitHub indicates that your pull request can be merged automatically follow the instructions in Appendix A at the end of this activity sheet and then return to this question when you have a pull request that cannot be merged automatically.

a. Give a screenshot from your pull request on GitHub showing that it cannot be merged automatically.

b. Briefly explain in a sentence or two of your own words, what happened that made your pull request go from being able to be merged automatically to now not being able to be merged automatically. Hint: Think Figure 1!

7. Use what you learned in the prior activity to synch the main branch of your local and origin repos with the upstream. Don’t forget to switch to your main branch first!

a. Give a screenshot of the commands that you used and their output.

b. Check that you are now synchronized:

* Check that the main branch on your origin repo on GitHub “is up to date…” (See #5).
* Use git status to check that the main branch in your local repo is up to date with your origin.

If the main branch in your local or origin repos are not up to date, double check your synchronization commands in part a and try again.

Nothing is required here, but the main branches in your local and origin repos must be up to date before continuing.

**Merge main Branch into Feature Branch:**

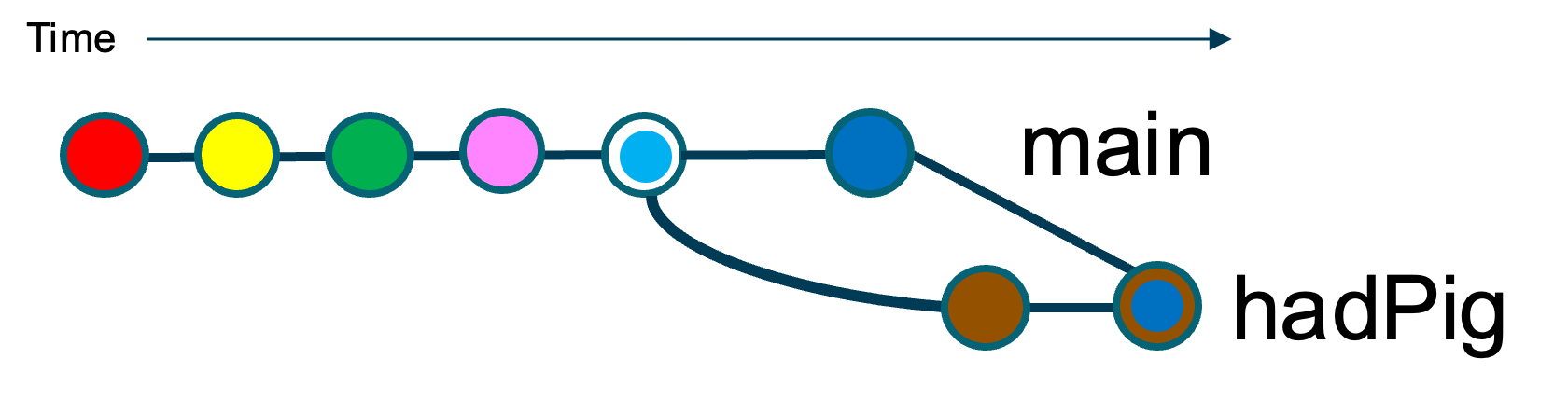
Your local main branch now contains the changes that were made to the upstream main branch. The next step is to merge the changes from the main branch into your feature branch. This is like the example from class, which is illustrated in Figure 2.

Figure - Merge main into feature branch

8. If you are going to merge the changes from the main branch into your feature branch, which branches will be the source branches? Which branch will be the target branch?

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | **Source Branches** |  |  |
|  | **Target Branch** |  |  |
|  |  |  |  |

9. The git merge <branch name> command will attempt to merge the changes in the active branch with the changes in <branch name> (i.e. the source branches) into a merge commit in the currently active branch (i.e. the target branch).

a. To use the merge command as just described your feature branch (i.e. the target branch) must be the active branch. Switch to your feature branch and confirm that it is now the active branch.

Give a screenshot of the commands you used and their output.

b. Use the merge command to try to merge the changes contained in the main branch and in your feature branch into your feature branch.

Give a screenshot of the command you used and its output.

10. The output from #9 should indicate that the automatic merge failed due to a conflict. If your merge command did not fail due to a conflict, return to question #6 and follow the instructions from there again.

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 contains the *raw conflict information* and copy and paste it here. Be sure to include the chevrons at the top and bottom of the conflict information.

11. Answer the following questions by highlighting the output that you gave in question #10.b.

a. Highlight the content of the Best Common Ancestor in gray.

Nothing is required here, just ensure that you have properly highlighted the output that you gave in question #10b.

b. Highlight the content showing the changes that the maintainers merged into the main branch after you made your feature branch in blue.

Nothing is required here, just ensure that you have properly highlighted the output that you gave in question #10b.

c. Highlight the content showing your changes in green.

Nothing is required here, just ensure that you have properly highlighted the output that you gave in question #10b.

12. Sometimes when you try to merge, and it fails due to a conflict you may want to undo the attempted merge. The git merge --abort command will undo a failed merge.

Undo the merge that you started in question #9 and use the cat command to display the file that contains the conflict again.

Briefly explain how you can tell from the output of cat that the failed merge has been undone?

**Resolving a Merge Conflict:**



Figure - Resolving a Merge Conflict

This section will walk you through the process of resolving the merge conflict using a graphical merge tool. The flow of this process is illustrated in Figure 3. You will switch to your feature branch, perform the merge, use the graphical merge tool to resolve the resulting conflict, and then stage and commit the merged content.

13. Ensure that you are on your feature branch. Then issue the command to merge the main branch into your feature branch (like you did in #9). The merge should once again fail because of the conflict. Use the cat command to confirm that the raw merge conflict information now appears in the conflicted file again.

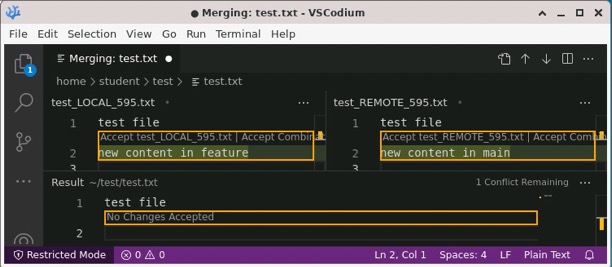
No answer is necessary here, but you need to ensure that the conflicted file contains the raw merge conflict information before continuing.

As you have seen when a merge fails git places the raw merge conflict information into the conflicted files. You could simply edit that file to resolve the merge conflict. However, in practice it is usually easier to use a *mergetool*. The KitClient you are using has been configured so that the git mergetool command will launch the the VSCode graphical merge tool, which you can use to resolve the conflict.

14. The git mergetool command will launch the VSCode merge tool so that you can use it to resolve the conflict.

a. Issue the git mergetool command.

The first time you issue this command VSCode will open the merge tool with a window containing three panes (LOCAL, REMOTE and Result). The content that you see will be different, but the window that you see should be similar to the one shown below:

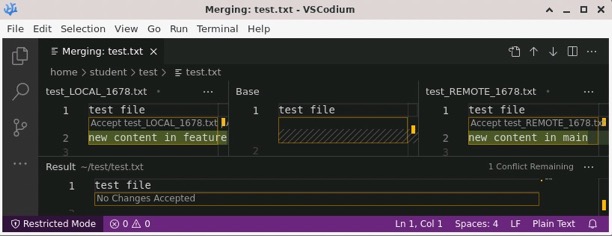


No answer is required here, but you must see a window similar to the one above before continuing.

b. Changing a few of the default settings will make the use of the merge tool easier to understand by displaying the Best Common Ancestor and both non-conflicting and conflicting changes.

Use the “…” menu in the upper right of the merge tool window and check the “” and “” options as shown to the right.

With these options enabled, the merge tool window should now contain 4 panes (LOCAL, Base, REMOTE and Result) and look similar to the one shown below:

****

No answer is required here, but you must see a window similar to the one above before continuing.

c. Paste a screen capture of your merge tool window showing the four panes and their content before making any changes.

d. Using your answer to part c, compare the Result pane to the Best Common Ancestor. How are they different? Why?

15. When using the merge tool, you resolve conflicts by modifying the Result pane so that it appears as desired. You can use the point-and-click “Accept …” and “Remove …” options provided by the merge tool to move content into or out of the Result pane or you can edit the Result pane directly.

Use the merge tool to resolve the conflict so that the Result contains:

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

Paste a screen shot of the merge tool window showing your final merged result here.

16. Save your changes in the merge tool and close the merge tool window.

Then issue a git status command.

a. Give a screenshot of your command and its output. Be sure that your output shows the names of any files containing changes that need to be committed.

b. Briefly explain why the output you see in part a makes sense based on what you have done.

17. From #16 you can see that you now have uncommitted changes. Those changes are all of the changes you made to the Result panel when performing the merge in the merge tool.

Commit the changes you made in resolving the conflict. Be sure to use a meaningful commit message.

Give a screenshot of the commands that you used and their output.

**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. Give the command you used and its output here.

19. Now visit the upstream repo and find your pull request. You should see that it can now be merged automatically.

If your PR cannot be merged automatically you have not correctly resolved the merge conflict, return to question #8 and try again.

Give a screenshot from your pull request showing that it can now be merged automatically.

20. Give the URL of your pull request.

**Reflection and Summary:**

21. 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 source branches into a target branch |  |  |
|  | Undo a merge that failed due to conflicts |  |  |
|  | Launch the configured graphical merge tool |  |  |
|  |  |  |  |

**More Practice:**

22. 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. The upstream repository contains a branch created just for this practice. Use the following command to get this feature branch from the upstream into your local repo:

git fetch upstream merge-conflict-practice

The merge-conflict-practice branch was created so that it contains some conflicts with the upstream main branch. When you have fetched this branch, think of it as if you had created it from main, then made some changes that resulted in conflicts with the upstream main branch. This leaves you at the point where you are trying to merge main into this feature branch and resolve the conflicts (e.g. like in Figure 3).

b. Make the merge-conflict-practice branch active and try to merge main into it. Give the commands that you used and their output here.

c. The merge-conflict-practice branch contains a few conflicts so the merge you attempted in part b should fail. Resolve the conflicts by accepting the changes from the merge-conflict-practice branch and all of the non-conflicting changes from the main branch.

Nothing is required here, but you must resolve the conflicts before continuing.

d. Save your changes and commit them to resolve the conflicts to your feature branch. Be sure to use a meaningful commit message. Give the commands that you used and their output here.

d. Push the merge-conflict-practice branch to your origin and create a pull request to the upstream. Give a link to your pull request.

**Appendix A:**

Skip this section if you were not directed here from question #6.

If you were directed here from question #6, then the pull request you created for your Round2 issue is showing that it can be merged automatically. The most common reason that this happens is that you synchronized your local and origin main branches after the instructor merged the conflicting changes into the upstream main branch. This could have happened if you did not complete the previous assignment on time. The instructions in this section will help you to resolve that issue but will require a little additional work on your part.

A.1. Ensure that you are in your local repository in a Terminal. Use the following commands to restore your main branch to the state before the conflicts were introduced:

git switch main

git reset --hard~1

You should see output similar to:

HEAD is now at …

A.2. You will now need to:

* Repeat the steps you used to fix your Round2 issue:
  + make a new feature branch
  + switch to your feature branch
  + make the changes necessary to address your Round2 issue
  + stage and commit your changes to your feature branch
  + Push your feature branch to your origin
  + Make a pull request to the upstream for your new feature branch
    - When you look at this pull request it should indicate that it cannot be merged automatically.

A.3. Return to question #6 and continue with this assignment.

**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)>