**08 – Version Control & Merge Conflicts**

**Activities**

COMP190 – Tools and Techniques for Software Development

Dickinson College

Fall 2020

Prof. Grant Braught

In the previous set of activities, you were introduced to a git branching workflow. For each code change (bug fix, new feature, etc.) you created a *feature branch* in your local repository and edited the code in that branch. To integrate that code into the main project (i.e. the *upstream*) you pushed the branch to your *origin* and then issued a *pull request* to the upstream. This is the foundation for how all team projects are managed using git and GitHub (or GitLab, or Bitbucket, etc).

This set of activities now considers the situation where many developers are working on the same project. Thus, while you are working on changes in your feature branch, others are also working on their own feature branches. When this happens, it is possible that multiple programmers are changing the same code at the same time. When one of these changes is merged into the main branch of the upstream it will result in a *merge conflict*. The activities below will walk you through a scenario that results in a merge conflict and the process of using a *merge tool* to resolve it.

**Setup:**

1. Before getting into the activities for today, you will want to do the following:

a. Launch your LinuxLite installation in VirtualBox.

b. Open a terminal.

c. Change into the directory containing your local Calculator repo from A07.

d. Checkout the main branch. What command did you use?

e. What output does git status generate?

f. Do you have the main branch checked out? How can you tell?

**Keeping Main in Synch:**

In the prior set of activities each of you had identified, fixed and created pull requests for at least two bugs in the Calculator project. The project managers thank you for your help and they have now merged these pull requests into the upstream main branch and closed the corresponding tickets in the issue tracker. Note that there were often multiple pull requests for the same issue and thus only one was merged.

With this newly merged code in the upstream main branch your local and origin main branches are out of synch with the upstream main. This section will walk you through the process of getting your local and origin main branches back into synch with the upstream.

*Setting the Upstream:*

2. Recall that your local repository was cloned from your origin. Thus, it doesn’t actually know where the upstream repository is. So, before you can get the changes from the upstream it is necessary to tell your local repository where the upstream is located.

a. Visit the upstream repository on GitHub and get the URL of the code:

i. Click the green “Code” button:



ii. Check the URL to be sure it is for the upstream repository and not your origin.

iii. Click the clipboard icon to copy the repository URL.



iv. Paste that URL as your answer to this question.

b. Confirm that the URL you have in part a is the upstream by checking that it contains COMP190 and not your username. If so, go on. If not, go back to part a and be sure to get the URL of the upstream.

c. In the terminal in your LinuxLite where you have your local repository.

i. Change into the directory containing your local repository.

ii. Use the command shown below to set location the upstream repository.

git remote add upstream <Upstream Repo URL>

iii. Confirm that the upstream repo was added using the command:

git remote -v

The output should contain lines indicating that pushes to and pulls from the upstream will go to the COMP190 repository. Note: You should also see your origin listed as well and those lines will indicate that pushes to and pull from origin go to your GitHub account. If not try this question again starting from part a.

iv. Give the output of the git remote -v command as your answer to this question.

Note that the upstream is set for a repository will only need to be set once. After you have done this git will remember it.

*Pulling from the Upstream:*

Because pull requests have been merged into the upstream, the main branch in your local repository is out of synch with the main branch in the upstream. In the next question, you’ll first confirm this and then the question after that will have you bring them into synch.

2. In your browser, examine the code for the Calculator class in the upstream repository. You will see that a number of the bugs have been fixed. These were fixed by pull requests that were merged into the upstream (one may even have been yours!)

a. You will likely notice that the plus method has been fixed. Give the line of code that now appears in the body of the plus method.

b. In your LinuxLite, cat the Calculator.java file in the main branch of your local repository. What line of code appears in the body of the plus method?

c. The two lines from part a and b should be different. Why?

3. From question 2 is should be clear that your local main branch is out of synch with the upstream. To bring them into synch you need to *pull* the main branch from the upstream to your local repository. This pull will *merge* any changes that have been made in the upstream main into your local main.

a. Use the command below to pull the main branch of the upstream and merge it with the main branch in your local repository:

git pull upstream main

b. What output was generated by the command in part a?

c. Now, cat the Calculator.java file in the main branch of your local repository again. What line of code now appears in the body of the plus method?

d. You should notice that the Calculator.java file in your local main branch is now identical to the one in the upstream repository. If it is not, ensure that you have checked out main branch and revisit this question starting from a.

*Pushing to Origin:*

At this point, the main branch of your local repository is synchronized with the upstream.

4. In your browser, go to your origin repository and examine the code for the Calculator class.

a. Give the line of code that appears in the body of the plus method.

b. Is the main branch in your origin in synch with your local and upstream repositories?

5. From question 4 it should be clear that the main branch of your origin repository is still not synchronized with the upstream. To synchronize your origin you need to push the main branch of your local repository to your origin. This was something you did several times for feature branches in the previous set of activities. Pushing the main branch is very similar.

a. Push the main branch of your local repository to your origin. What command did you use?

b. Confirm that the main branch on the origin is now in synch with both your local and your upstream repositories. Briefly explain how you confirmed this.

The process of keeping main in synch (i.e. pulling the main branch from the upstream and pushing it to origin) is something you will do often. So, it won’t be long before you remember those commands and have them at your fingertips.

**Deleting a Feature Branch:**

At this point the bug fixes from your feature branches have been integrated into the upstream through your pull request (or possibly someone else’s). Those changes are now also contained in the main branch of both your local and origin repositories. Thus, there is no need to retain your feature branches. Note: Some programmers are inclined to retain old feature branches for personal reasons. But if you work on a project for a while having a large number of old feature branches around creates clutter. Additionally, your GitHub account and the upstream will have a complete record of the pull request, so you can see easily see what it contained at any time. Thus, it is common practice delete feature branches after the changes they contain have been merged.

6. What branches currently exist in your local repository? Give the output of the git branch -a command as your answer. Recall that this command lists all of the branches in your repository.

7. What branches currently exist in your origin repository? In your browser go to your origin repository on GitHub and click the A picture containing graphical user interface

Description automatically generatedbutton to display the list of branches.

8. Deleting a feature branch is a two-step process. The first step deletes the branch from your local repository. The second step pushes that change to your origin repository.

a. Use the command below to delete the one of your feature branches:

git branch -D <branch name>

b. Use the command that lists all of the branches to confirm that the branch has been deleted. Give the output of the command here. If the branch has not been deleted, revisit part a.

c. Use the command below to push this change to your origin (note the : in the command below, it is important!):

git push origin :<branch name>

d. In your browser go to your origin repository on GitHub and confirm that the branch has been deleted there as well. If not revisit part c.

e. Delete your other feature branch and give the two commands that you used. Be sure to confirm that your branches have been deleted both in your local repository and in your origin repository.

**Working on a New Issue:**

All of your repositories are now back in synch with the upstream and you are ready to begin working on the next feature or bug fix that you can contribute to the project. As you work on this issue some of the steps you complete will create a conflict between your changes and changes integrated into the upstream. In practice, these changes would have been added to the upstream via pull requests by others who are contributing to the project. For these activities we will simulate the changes to the upstream.

10. Visit the issue tracker and find issue #22 that asks for the sphVol method to be renamed. What should the new name of this method be? Again, recall that these are intentionally simple changes so that we can focus on learning about git/GitHub.

11. Create and checkout a new feature branch for your work on this issue. What commands did you use?

12. Use your editor of choice and modify the name of the function that computes the volume of the sphere to be as requested by issue #22 in the issue tracker.

13. Stage your change and commit it to your local repository with a meaningful commit message. What commands did you use?

**Merging the Main Branch:**

Once you complete a change you need to synchronize your main branch with the upstream. You can then merge any changes that have been made in the upstream into your feature branch to ensure that it is compatible with the current upstream. This will make it possible for the project maintainers to easily merge your pull request.

For logistical reasons we will be simulating both changes made to the upstream and the process of you synchronizing with it. In practice, you would synchronize with the upstream main just as you had in earlier in questions 3 and 5.

*A Merge Without a Conflict:*

Imagine now that while you were working on your fix for the sphVol method name, someone else was working on another issue. Perhaps, issue #21 that asks for the cubeVol method to be renamed as cubeVolume. Also imagine that they completed their work, submitted a pull request and that the project maintainers merged their change. Finally, we will assume you have synchronized the main branch of your local and origin repositories with the upstream, so that it now contains their changes.

14. Issue the following sequence of commands to simulate the things that were just imagined:

git checkout main

﻿sed -i 's/cubeVol/cubeVolume/g' Calculator.java

git add Calculator.java

git commit -m “Faked Pull Request”

git push origin main

The cubeVol method in your main branch has now been renamed cubeVolume, as if someone had merged it into the upstream before you synchronized.

15. cat the Calculator.java file to confirm that this method has been renamed. If it has not, revisit question 14.

16. The git diff command is useful for displaying the differences that exist between two branches. For example, git diff main fixSphere will show you the differences that exist between the fixSphere branch and the main branch.

a. Use git dif main <branch> to compare the main branch to your feature branch. What output is generated?

b. The output of git diff shows you the lines that would have to be added to (+ , green) or removed from (-, red) one branch to make it the same as the other. Study the output generate in part a. Would the changes indicated convert the main branch into the feature branch or the feature branch into the main branch? Briefly justify your answer.

Because the changes that were merged into main (by our fake pull request) and the changes you have made in your feature branch affect different parts of the code they do not create a conflict. When changes do not create a conflict, git is able to merge them automatically.

17. To merge the changes from main into your feature branch:

a. Checkout your feature branch.

b. Perform the merge using the following command:

git merge main

c. When you run the above command, git will open the nano text editor and ask you to write a commit message for the merge. In practice, you should write a meaningful commit message for the merge. For now, the default message will be sufficient. Use the ctrl-X keys to exit nano and accept the default message.

d. What output is generated when you perform the merge?

e. Use cat Calculator.java to confirm that your feature branch now contains the changes to both the cubeVol and sphVol method names. If not, revisit questions 16 and 17.

18. Your feature branch is now synchronized with the upstream and is ready for you to make a pull request for the fix. Push your feature branch to your origin and make a pull request for this fix. Your pull request on GitHub will be your answer for this question.

*A Merge with a Conflict:*

During the previous synchronization with the upstream and merge there were changes to be merged, but these changes did not create conflicts. This allowed git to perform the merge automatically. You’ll now work through an example where there are merge conflicts that prevent git from performing the merge automatically. These merge conflicts will require you to intervene to specify how they should be resolved.

19. Visit the issue tracker and find issue #19. What is the title of this issue?

20. Fix issue #19. Be sure to follow the branching workflow:

a. Create a feature branch. What command did you use?

b. Checkout the feature branch. What command did you use?

c. Complete the fix in the feature branch.

d. Stage your changes.

e. Commit your changes with a meaningful commit message.

List the commands that you used.

21. Issue the following sequence of commands to simulate synchronizing with the upstream main that has had changes merged into it:

git checkout main

﻿sed -i 's/cirAr/areaOfCircle/g' Calculator.java

git add Calculator.java

git commit -m “Faked Pull Request”

git push origin main

The cirAr method in your main branch has now been renamed areaOfCircle, as if someone else had submitted a pull request for that change and then it had been merged into the upstream before you synchronized.

22. Briefly explain why the changes made to the main branch in question 21 will create a merge conflict.

23. Use git diff to display the changes from main to your feature branch.

a. What is the full command that you used (Hint: see question 16)?

b. What output was generated?

24. Attempt to merge the main branch into your feature branch.

a. What is the full command that you used?

b. What output did it generate?

c. The output should indicate that there was a merge conflict and that the automatic merge failed. If it does not revisit questions 20-24.

25. When there is merge conflict that cannot be resolved automatically git will insert information about the conflict into the source code files. Open the Calculator.java file using your preferred editor. Copy the lines beginning between the ﻿<<<<<<< HEAD and the ﻿>>>>>>> main and paste them below.

26. These lines describe the conflict. The code just below HEAD is what appears in your feature branch. The code just above main is what appears in the main branch. It is the conflict between these lines that git is unable to resolve. You can resolve the merge conflict by editing these lines back into valid Java that combines the two possibilities in the desired way. Edit the file so that it contains just the changes that you made in the feature branch and save the file. There is nothing to turn in for this question, it will be checked using the pull request you create below.

27. You have now edited the Calculator.java file. So you need to stage and commit it. What commands did you use?

28. Push your feature branch to your origin and create a pull request to the upstream for this fix.

It is not required viewing, but if you’d like another take on merge conflicts and resolving them manually the video *Resolving Merge Conflicts* by The Coding Train does a nice job:

* <https://www.youtube.com/watch?v=JtIX3HJKwfo> (8:48)

**Resolving Conflicts Using a Merge Tool:**

Resolving merge conflicts by manually editing the files is relatively easy when the conflicts are small and easy to find. When they become more complex, touching multiple locations in a single file and possibly spread across multiple files a *merge tool* can help. A merge tool is a graphical tool that displays merge conflicts in a way that makes them easier to see and resolve.

*Setting up the Merge Tool:*

29. There are a wide range of different merge tools available and many developers will have a favorite. We’ll look at a merge tool named meld here, most others provide similar functionality.

a. Use the synaptic package manager to install the Meld merge tool on your LinuxLite virtual machine. Use the search feature in the package manger to find the appropriate package. What is the full name of the meld package?

b. Run meld from the command line to ensure that it works. You can quit the Meld application once its window appears and you know that it works. If you receive an error trying to run meld from the command line revisit part a.

30. A little configuration is necessary to tell git that it should use Meld as the merge too. Enter each of the following commands in the terminal:

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

Give the full output of the command git config --global --list as your answer to this question.

*Resolving a Conflict with a Merge Tool:*

31. Visit the issue tracker and find issue #15. What is the title of this issue?

32. Fix issue #15. Be sure to follow the branching workflow:

a. Create a feature branch. What command did you use?

b. Checkout the feature branch. What command did you use?

c. Complete the fix in the feature branch.

d. Stage your changes.

e. Commit your changes with a meaningful commit message.

List the commands that you used.

33. Issue the following sequence of commands to simulate synchronizing with the upstream main that has had changes merged into it:

git checkout main

sed -i 's/div/doDivision/g' Calculator.java

git add Calculator.java

git commit -m “Faked Pull Request”

git push origin main

The div method in your main branch has now been renamed doDivision, as if someone else had submitted a pull request for that change and then it had been merged into the upstream before you synchronized.

34. Attempt to merge the main branch into your feature branch.

a. What is the full command that you used?

b. What output did it generate?

c. The output should indicate that there was a merge conflict and that the automatic merge failed. If it does not revisit questions 32-34.

35. Run the command git mergetool. When you do, you will see a window like the one shown below:



As described in the video, the code on the left is your feature branch and the code on the right is the main branch. Your job with the merge tool is to make the code in the center pane appear as you would like it to by choosing your code, the code from main, or editing it into some combination of the two.

a. Play around with the meld application a bit. Click the arrows to see what they do. Edit the code in the middle panel directly. But, ultimately, make the code in the middle pane the same as the code from your feature branch.

b. Choose “Save” to save the code in the middle pane into the Calculator.java file.

c. Quit the Meld program.

d. Run the git status command. What output is generated?

e. The output in part d indicates that you now need to commit your changes (Note: The changes are staged automatically when you run git mergetool). Use git commit with a meaningful commit message to commit your changes to your feature branch.

36. Push your feature branch to your origin and create a pull request to the upstream for this fix.

It is not required viewing, but if you’d like another take on merge conflicts and resolving them using a merge tool the video *Git Tutorial: Diff and Merge Tools* by Corey Schafer does a nice job. Note that his video uses the tool diffmerge instead of Meld, but they are quite similar (the left and right panels are just the reverse of what they are in Meld).

* <https://www.youtube.com/watch?v=iCGrKFH2oeo> (9:37)

**Review & Reflection:**

37. Paste your list of git commands from activity A07 here and add to them a list of all of the new git commands that were used in this set of activities. Include one or two sentences describing what each does.