**Tutorial 5**

***Resolving Merge Conflicts***

Sometimes a *merge* will cause a conflict. For example, suppose someone changes a file in in *master* and then someone changes the same thing in a new branch. When the new branch is merged with *master* there will be a conflict and Git cannot accomplish the merge. In this case, you must resolve the conflict manually, stage and commit the resolution. In this tutorial we consider a simple case of this and how to resolve it.

A conflict will exist anytime a line of code in one branch is modified (or deleted), while that same line in another branch is modified. However, note that this does not occur in the simple case we had in the previous lab. There, we branched off of *master,* made a change in the branch, and then merged back – all the while *master* had not been changed. This is called a *fast-forward merge.*

The scenario for this lab is:

1. You have a working piece of code in the *master* branch. You want to start development on a new feature so you create a new branch, *newfeature2*. You work on the code in *newfeature2*.
2. Suddenly an urgent change needs to be made to the code base (*master*). You switch back to the *master* branch, create a new branch, *quickfix*. You make the change, stage, commit, and the merge back into *master.* This will be a *fast-forward* merge.
3. Then, you go back to the *newfeature2* branch, continue working making a change that will conflict with a change made in *quickfix* that has now been merged into *master.* The resulting merge has a conflict which we must be resolved.

This lab continues from Tutorial 4.

**Steps to Complete – Part 1**

This shows an example of a [Competing line change merge conflicts](https://help.github.com/articles/resolving-a-merge-conflict-using-the-command-line/) and is similar to the steps below.

1. Create a new branch and check it out:

**λ**  git checkout –b newfeature2

1. Open *foo.txt* and change, “Hello World” to “Hello YourFirstName YourLastName” (make sure and supply your name), save and close.

**λ**  notepad foo.txt

1. Stage and commit changes to *newfeature2* branch.

**λ**  git commit –a -m "Changed to YourFirstName YourLastName"

1. (Read, no action required) *An urgent message has just come in that we need a quick fix to master…*
2. Switch back to the *master* branch, create a new branch, *quickfix* and switch to it:

**λ**  git checkout master

**λ**  git checkout –b quickfix

1. Open *foo.txt* and change, “Hello World” to “Hello you people”, save and close.

**λ**  notepad foo.txt

1. Stage and commit changes to *quickfix* branch.

**λ**  git commit -a -m "Changed to you people"

1. Switch back to the *master* branch and merge the *quickfix:*

**λ**  git checkout master

**λ**  git merge quickfix

1. Display contents of *foo.txt* and verify that top line says, “Hello you people”.

**λ**  type foo.txt

1. Switch back to the *newfeature2* branch

**λ**  git checkout newfeature2

1. Open *bar.txt* and **change this text at the end: “More stuff in new file” to “Less stuff in new file”.** Save and close.

**λ**  notepad bar.txt

1. Stage and commit changes to *newfeature2* branch.

**λ**  git commit -a -m "Less stuff"

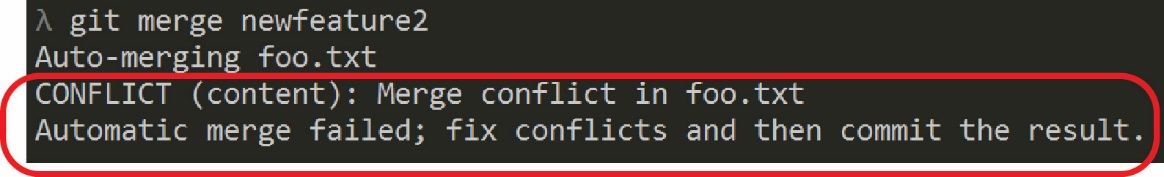
1. Switch back to the *master* branch

**λ**  git checkout master

1. Attempt to merge *newfeature2* with *Master*:

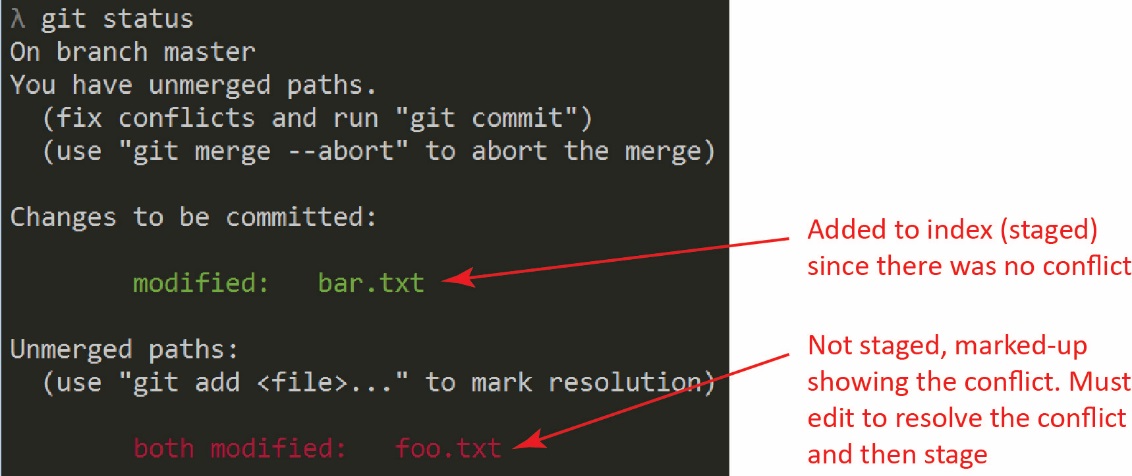
**λ**  git merge newfeature2

The result is shown below. The problem is that *foo.txt* in *Master* has, “Hello You People” and *foo.txt* in *newfeature2* has, “Hello Everyone”.



1. Check the status of *master.* The result is shown on the right.

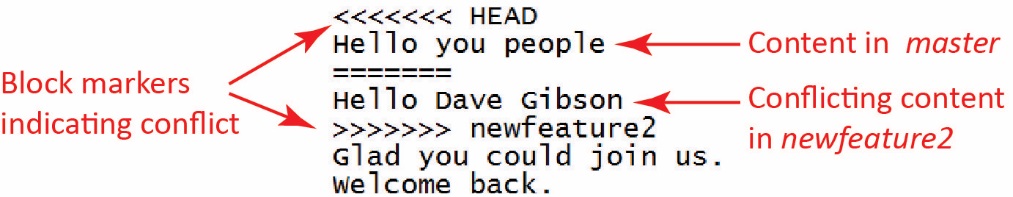
**λ**  git status



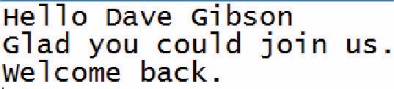
1. Open the *foo.txt* in notepad and you will see the conflict which has been marked-up by Git. Do not close the file, you will change it in the next step. Do the following:
2. Display *foo.txt*:

**λ**  notepad foo.txt

The result is shown below (there may also be the text, “Roundabout” that we added earlier):

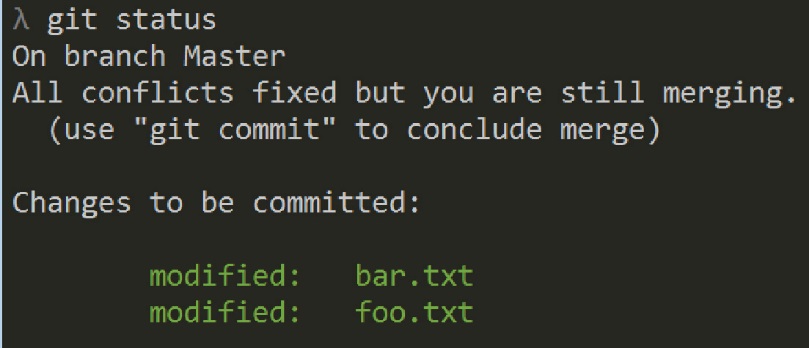


1. One way to resolve this conflict is to simply edit the file, replacing the markup Git has added (*i.e.* “<<<…”, *etc.*) and leaving the file as you desire it. Do this now: remove all markup leaving the text, “Hello YourName” as shown below. Then, save and close.



1. Note that the merge attempted earlier did not occur. Thus, we need to stage the change we just made to the marked up *foo.txt*:

**λ**  git add foo.txt

1. ****Display the status

**λ**  git status

Note: Git still thinks it is “merging”. When we commit these changes the merge will be complete.

1. Finally, commit the changes:

**λ**  git commit -m "Merge conflicts resolved"

1. Delete the two branches:

**λ**  git branch -d newfeature2

**λ**  git branch -d quickfix

**Steps to Complete – Part 2**

This portion of the tutorial shows an example of a [Removed file merge conflicts](https://help.github.com/articles/resolving-a-merge-conflict-using-the-command-line/) and is similar to the steps below.

1. (Read, no action required). Here we show an example of where one branch (say *master*) has changed a file while another branch has deleted it.
2. Create a new branch (but don’t activate it):

**λ**  git branch newfeature3

1. Open (in *master*) *bar.txt* and replace all existing text with: “Friday”, save and close.

**λ**  notepad bar.txt

1. Stage and commit:

**λ**  git commit -a –m “new bar in town”

1. Checkout *newfeatue3*

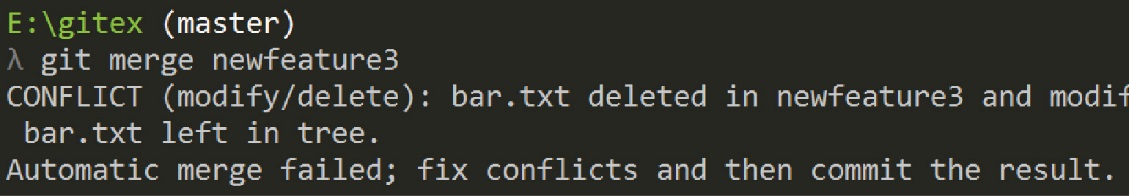
**λ**  git checkout newfeature3

1. Remove *bar.txt* (from *newfeature3*) (this will be the “old” one, without “Friday”)

**λ**  git rm bar.txt

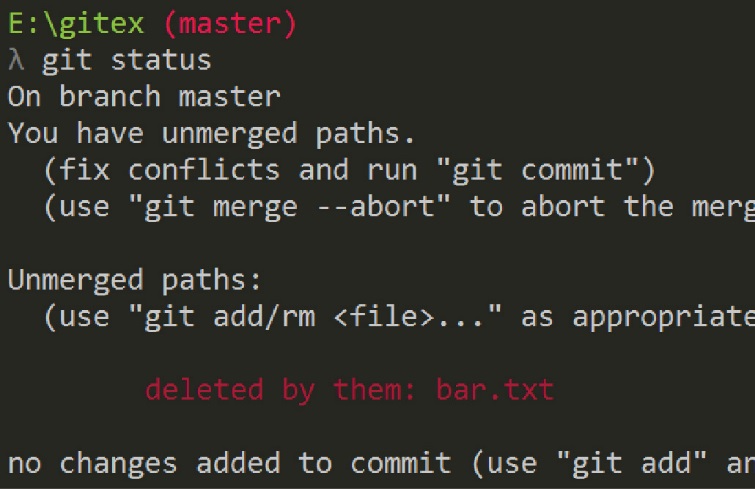
1. Stage and commit

**λ**  git commit -a -m "Really don't need any more"

1. Return to *master* and merge *newfeature3*. The result is shown on the right.

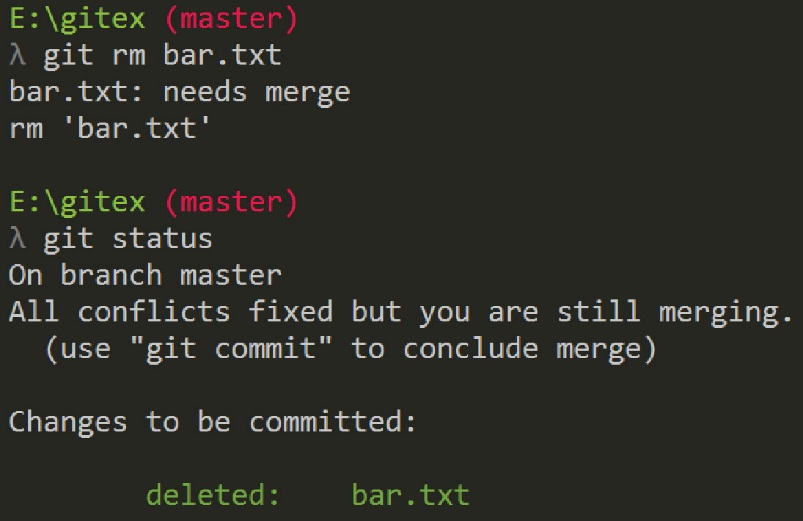
**λ**  git checkout master

**λ**  git merge newfeature3

1. Check the status. The result is shown on the right.

**λ**  git status

There are two choices here: remove the file or keep it. In the next step, will remove it.

1. Remove the file and stage it, then check status. The results are shown on the right

**λ**  git rm bar.txt

**λ**  git status

1. (Read, no action required) If we had wanted to keep *bar.txt*, we would have:

**λ**  git add bar.txt

and then committed the same as we do in the next step.

1. Finally, commit the deletion, check the status and note that *master* is clean

**λ**  git commit -m "Removed bar.txt via conflict resolution"

**λ**  git status

1. (Optional) It is likely as you work on your project you will need to learn more about branching, merging, and rebasing. More information about [Basic Branching and Merging](https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging). A merge that has conflicts can also be aborted as explained in [Advanced Merging](https://git-scm.com/book/en/v2/Git-Tools-Advanced-Merging#_advanced_merging). An alternative to merging is *rebasing* which we consider in the next tutorial*.*
2. **Do the following:**
3. Make a screen shot of the top 3 commits of: git log (in *master*). Make sure it shows your name and date.
4. Place the image in the *HW VCS* document in the appropriate place.
5. The image should be easily readable without zooming in or out.

Good idea: make a backup copy of your *gitex* folder with the name: *gitex\_5*. Then, you’ll start Lab 6 using the *gitex* repo.