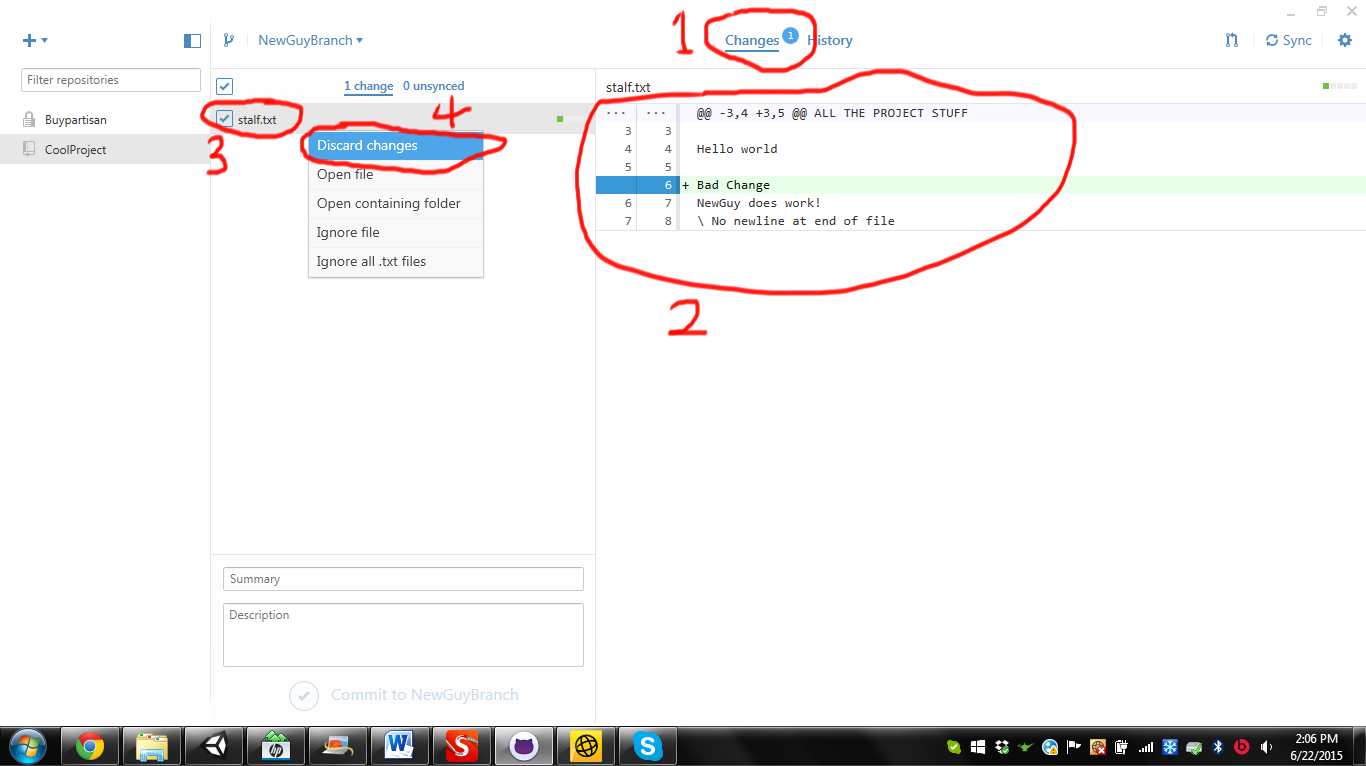
Merging and You

Sometimes bad things happen in projects and merge conflicts can occur. Merge conflicts happen when two or more people check out a file and try to check it in after having both altered it. In our project we have created a file checkout system which should help us to avoid merge conflicts but sometimes even that fails.

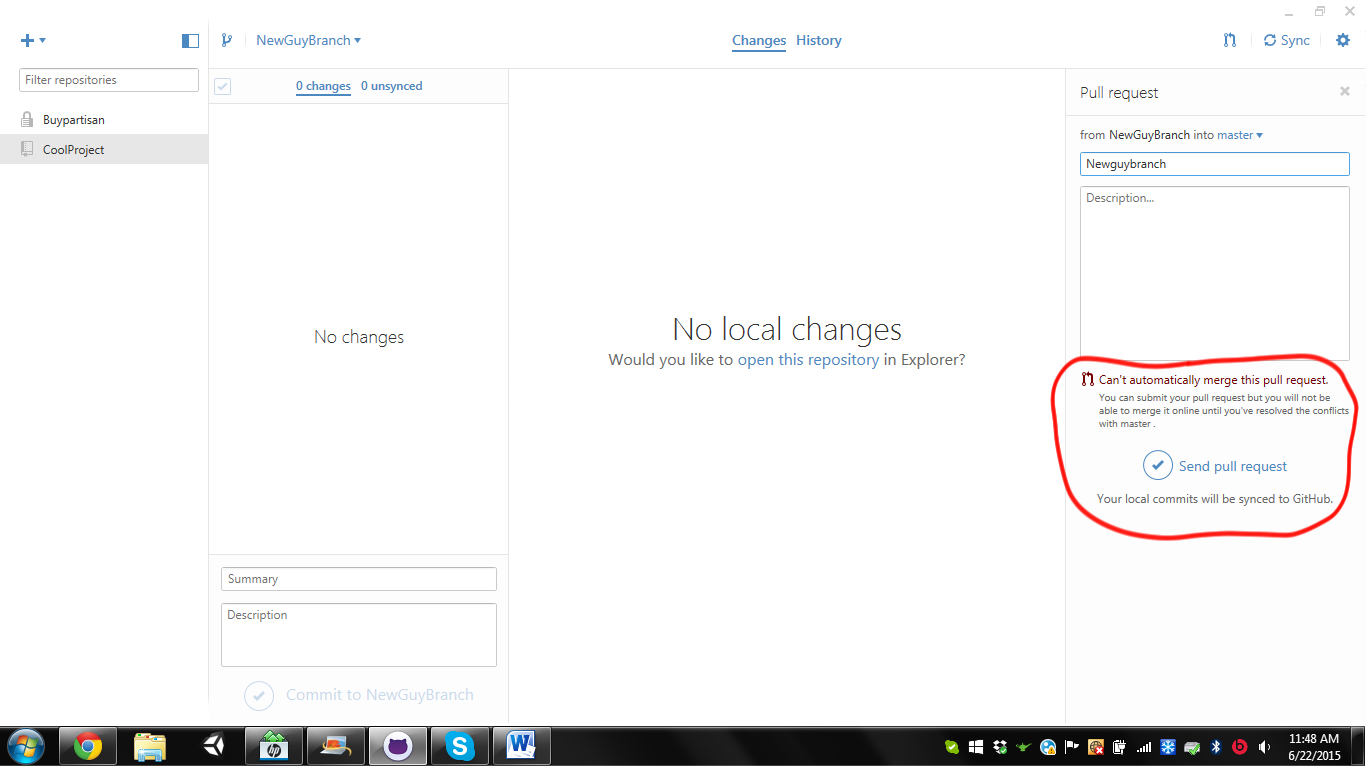
This tutorial will guide you through the process of dealing with a manual merge.

Before we get started if you have accidently made a change that you know will cause a merge conflict you can always get rid of the change by going to changes(1), finding the change in the files modified(2). Then right click the file in the change log(3). Finally click discard changes on the file you wish to discard the changes on(4). **This can only be done BEFORE you commit the changes so be sure to double check the files modified every time you commit!**



Let’s get started!

Suppose you are minding your own business and decide to work on a feature. You are coding along when all of a sudden when you go to merge and this message pops up!

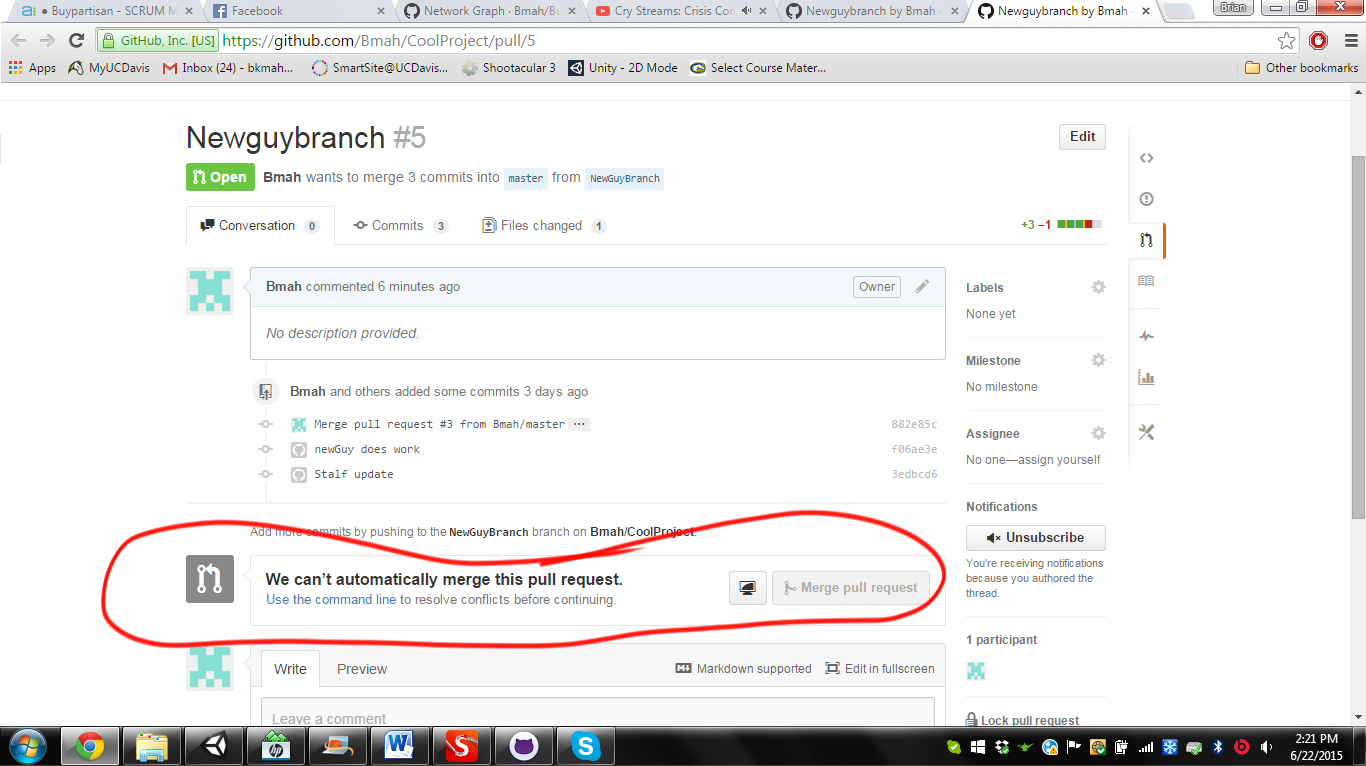


So it appears you have a merge conflict, let’s see how to deal with this situation!

Before you do anything **Backup your files**. Things can happen and stuff can get lost so be sure to make a backup before you start doing stuff.

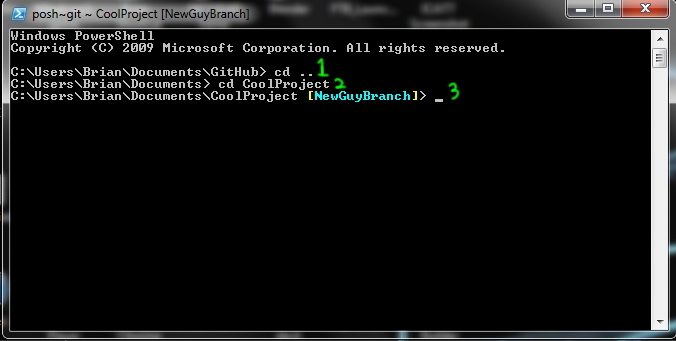
First write in a merge title and description. Be sure to comment that there was a merge conflict in the description. Click Send Pull request as per usual and lets head to GIThub!

You should see that the Merge pull request is now greyed out and that you cannot merge this request automatically.



To fix this you must manually choose which file to use. This is done in 2 ways depending on the kind of file that is causing the problem.

Go to your git shell(the console) and navigate to the project. The command “cd ..” will take you out a level like hitting the back button on a browser(1). The command “cd (path you want to go to)” will take you in a level like double clicking a folder(2). When you are at the project you will see something that looks like C:\FilePath\FilePath\ProjectName [branch ]> \_ (3).



Once you are in your working directory you should type the command: **git status**.

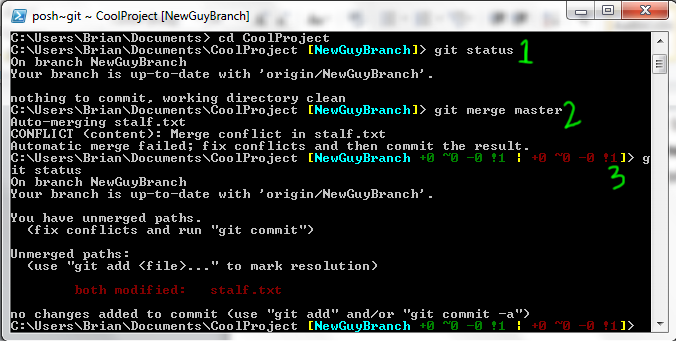
It should tell you if something is wrong like your current branch. If everything is fine then it will display:

“Your branch is up-to-date with ‘origin/YourBranchName’.

Nothing to commit, working directory clean”(1)

Now that we are sure everything is ok on our branch it is time to manually solve this merge conflict. Type the command: **git merge BranchYouWantToMergeWith**. This should tell you that you have a merge conflict.(2)

Finally type in: **git status**.(3)



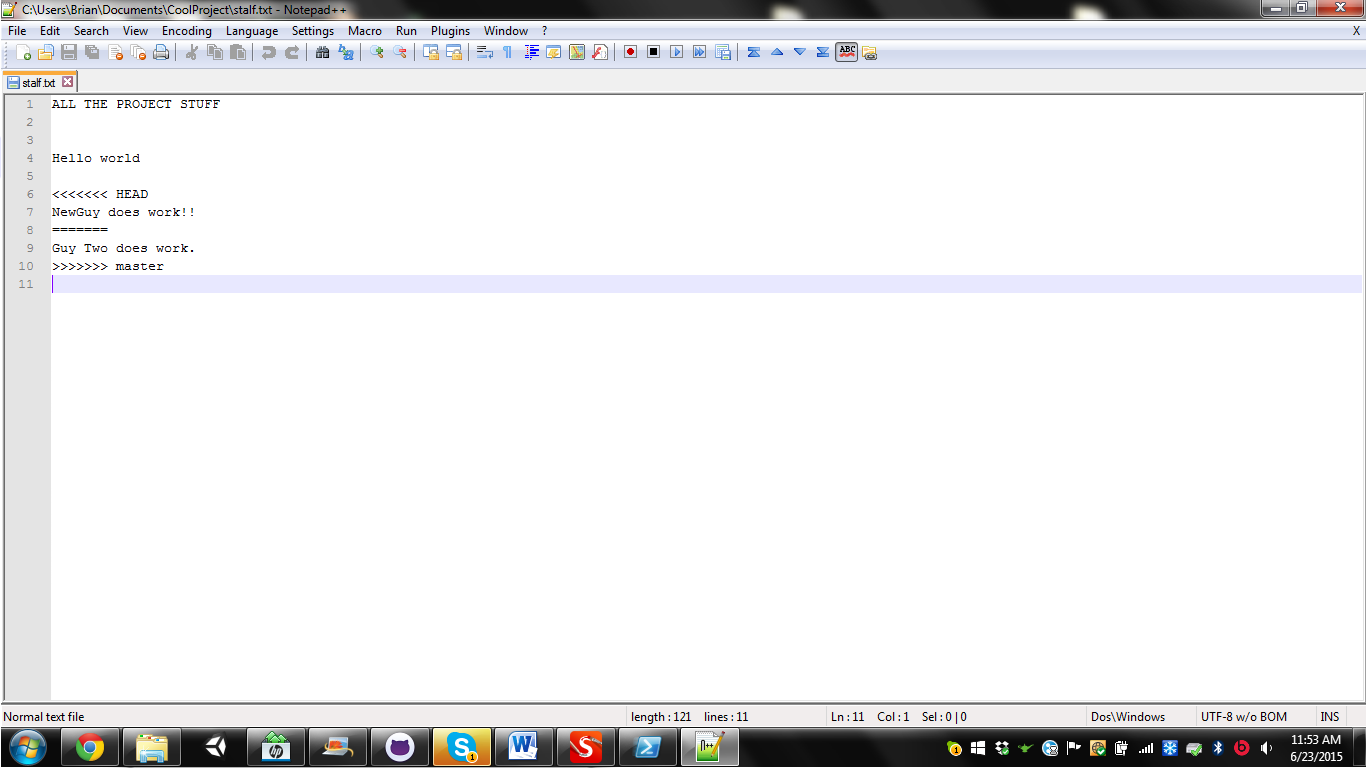
The files under “both modified:” are the files that are causing the merge conflict.

This is where the two kinds of merge conflicts come into play. Depending on the type of file the manner in which you solve the merge conflict will vary.

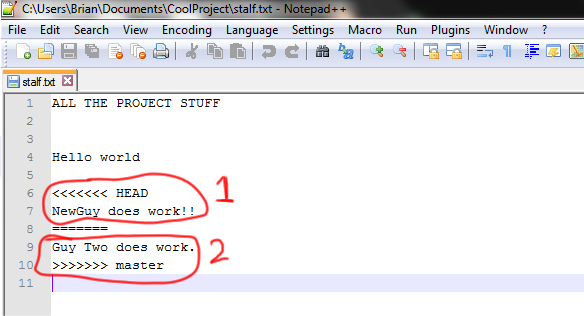
In this case it is a text file named stalf.txt

To solve a text file of any type (.cs, .txt, .cpp, ect) simply open the text file in a text editor of your choice. You should see that some alterations have been made to your file.

For my file I will be using Notepad++.



Let’s get a closer look at what is going on here:

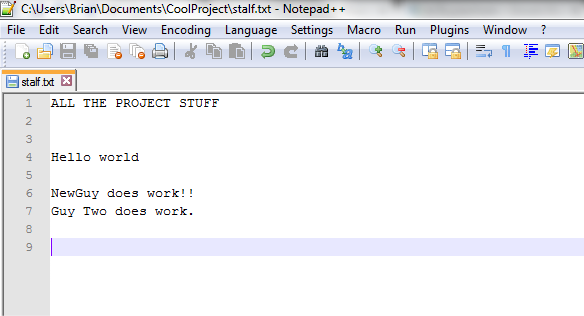


To start there is the line <<<<<<< HEAD followed by the line of code that you have pushed (1).

Then there is a line of ======= acting as a separator.

Finally there is the line of work that is conflicting with your code followed by >>>>>>> master (2).

You must manage the lines so that the text file is in the state you want it. In this case we want both lines of text to be there so I edited the file to look like this:



Also notice that in addition to combining the pieces of text that I wanted I also removed the <<<<<<< HEAD, =======, and >>>>>>> master lines.

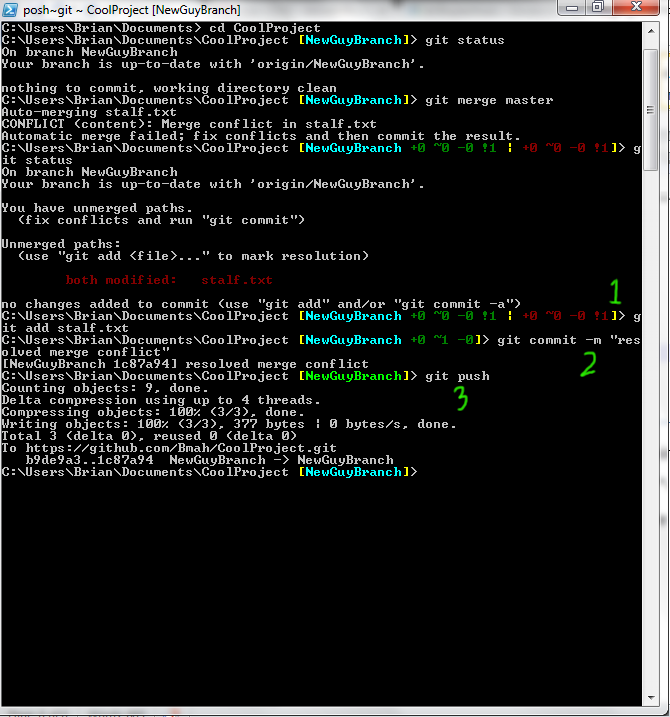
For any other kind of non-text file(.jpg, .unity, .png) you will not have the nice text differentiation that has both peoples changes. You will have to either push your changes or ask the other person for their files and push their changes. To have both parties changes one person will have to redo their work.

Once all the changes you wanted have been made save the project and go back to the GIT shell.

Add the file you changed with: **git add FileYouWantToAdd** (1)

Then commit the changes with : **git commit –m “Comment Describing The Merge”** (2)

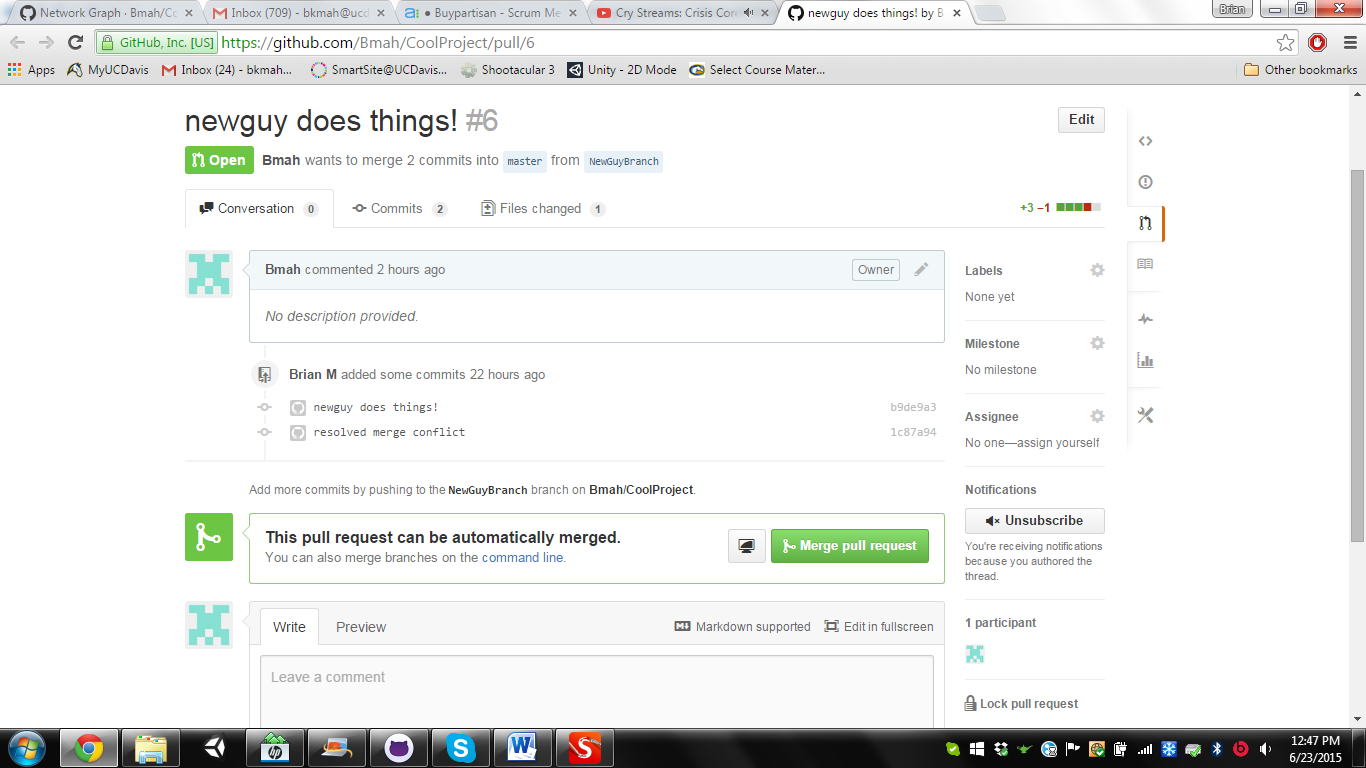
Finally push the changes onto your branch with: **git push** (3)



Once this is done close your GIT shell.

Go back to the merge request that you opened way back at the beginning.

You should now be able to merge the branch.



Now that you have gotten this far HIT THAT MERGE BUTTON! Everything should proceed as normal. Congratulations you have successfully resolved a merge conflict!