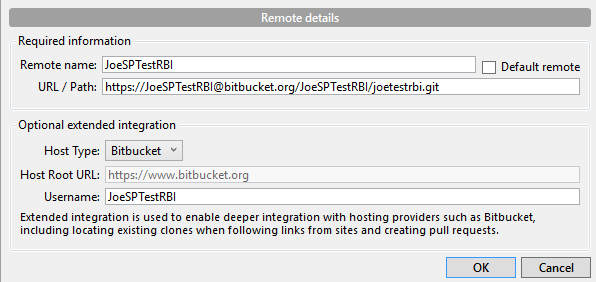
Git behaviour

Test repository set up



If you make an uncommitted change to a file within branch A, then make branch B the working branch, you will not overwrite those changes.

If you make a change to a file within branch A, commit the change to branch A, then make branch B the working branch, you will overwrite those changes. Making branch A your working branch will restore the change.

Create a branch but don’t push it. The branch isn’t created on remote. But that doesn’t make any difference re the above piece of behaviour.

Branch1 represents the current release, Branch2 represents the next release. Both have different changes to the same file. Periodically, Branch2 should be ‘refreshed’ from Branch1. Ie Branch1 should be merged with Branch2. This should leave the file in Branch1 unaffected. The file in Branch2 should contain Branch1 changes and Branch2 changes.

File on Branch1 should still look like the below:

File1 text is here  
New text  
Add a new new line in here  
A change I've made on Branch1  
A change I've made on Branch1a

File on Branch2 should look like the below, once the manual merge takes place:

File1 text is here

New text

Add a new new line in here

A change I've made on Branch1

A change I've made on Branch1a

Add a new line in here on master branch

Change made in Branch2

Another Change made in Branch2

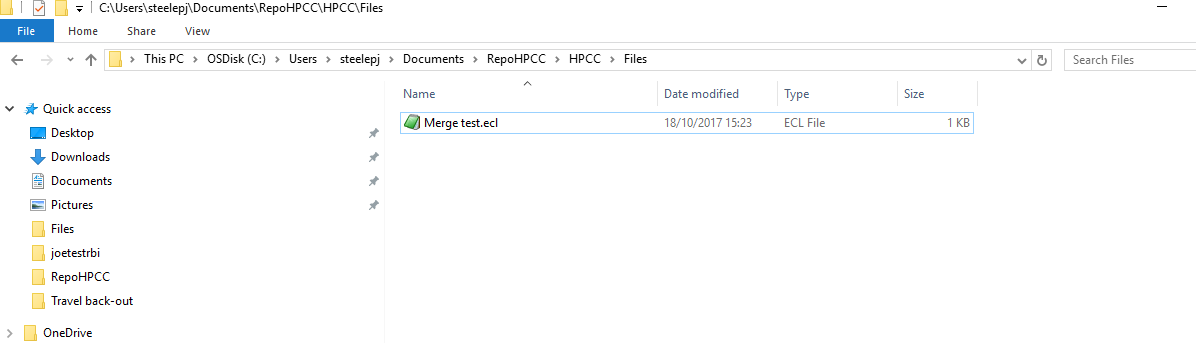
**Question**

I have a remote called JoeSPTestRBI which (I thought) was connected to this local repository

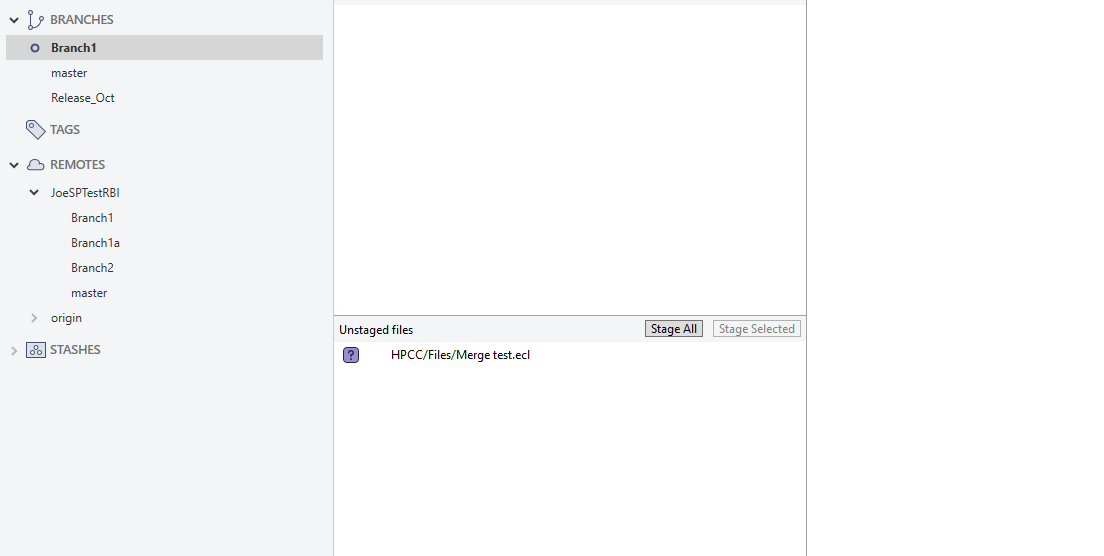
C:\Users\steelepj\Documents\joetestrbi

It should have nothing to do with C:\Users\steelepj\Documents\RepoHPCC\HPCC\Files.

But when I make it my working branch, I lose the contents of that file…

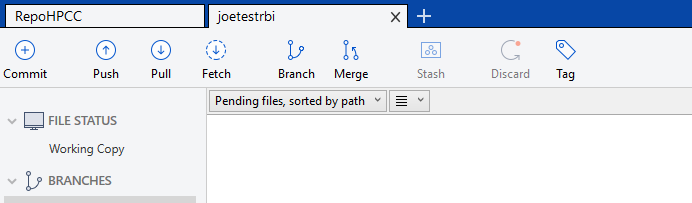


It also is looking in that folder for uncommitted changes?



Why can’t I keep the two repositories separate?

You do it by keeping them on separate tabs:



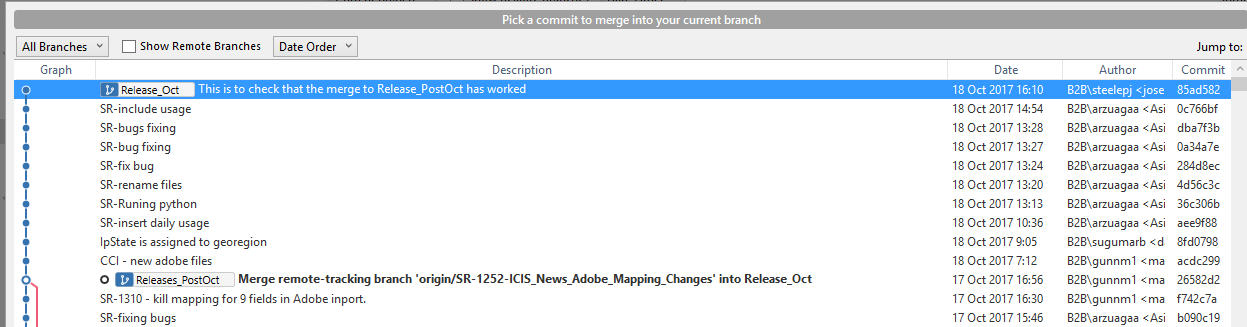
What does deleting a remote repository do? You lose the Repository in Repository Settings.

**How to merge**

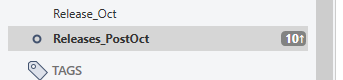
Make the branch that you want to merge into your working branch.

Click on the Merge button

Find the branch you want merge from and select the commit point you want to merge from and click OK. Note that this only merges locally.



The branch you are merging from should show there are commits to push.



Push them.

TEST

Create two new files and commit to the Master branch

Merge Master branch to Branch1 so they exist there as well

1 Change one of the files only on Branch1 and commit

2 Change the other file on Branch1 and commit

Merge Branch1 with Master but only at the point of the commit at 1 above

Only the first file is updated in Master

Change first file again and commit to Branch1

Again merge Branch1 with Master but this time select the latest commit

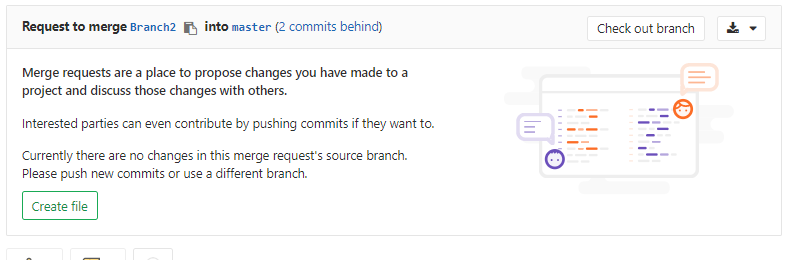
Both files have updated in Master

Add two more files to Branch1 and add as separate commits.

Something to Test

Does merging a branch every create conflicts? Yes, if a file has been updated since the source branch had been created then a conflict will be flagged

What does this mean?



It means that 2 commits have been made to the Master branch since the Branch2 was create from the Master branch. This may cause a conflict.

**Merging in GitLab**

**Test 1**

On the Master branch create a new file called MergeConflictFile.

Create a new branch from the Master branch called Feature\_Branch3

On this branch, make a change to MergeConflictFile.

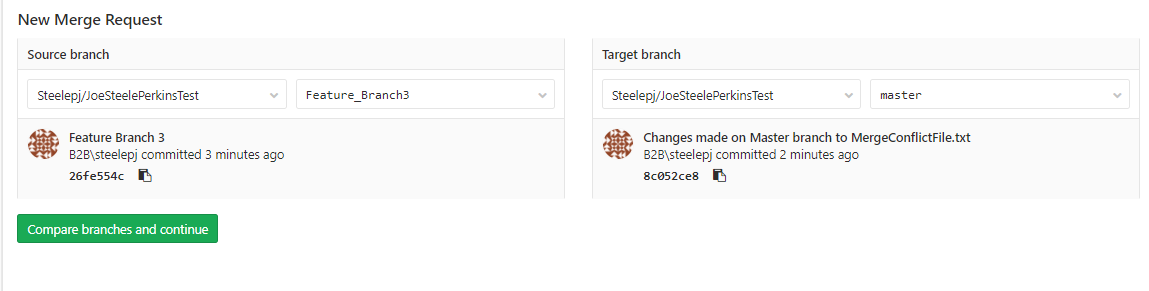
And create a new file called FeatureBranch3.

Back on the Master branch, make a change to MergeConflictFile.

Now we want to merge FeatureBranch3 back to Master. Should cause a conflict. We want to keep what is on FeatureBranch3.

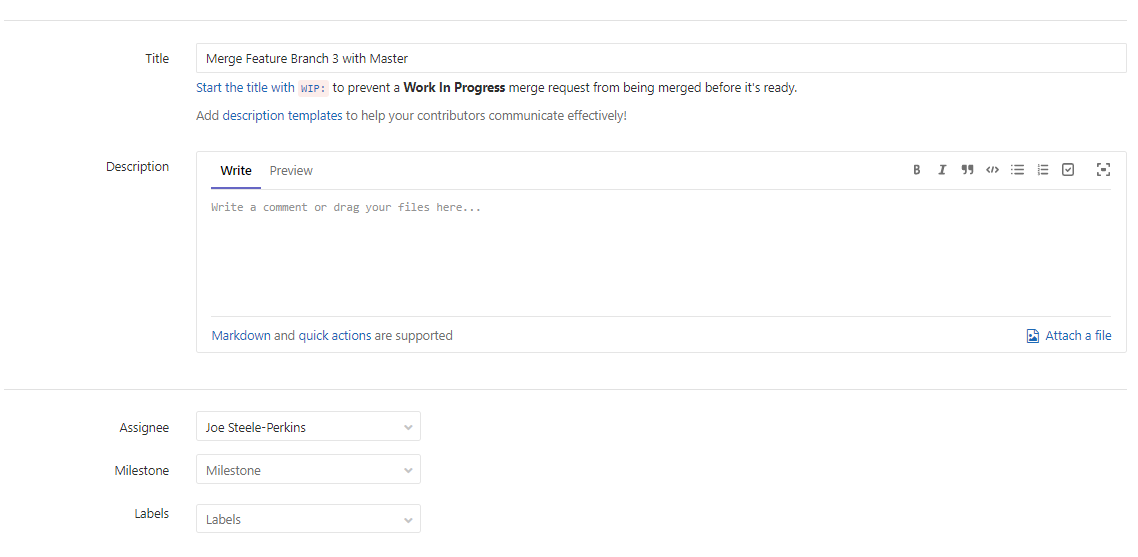
Create a new Merge Request

What does this show us? Shows when the last commit was made on each branch. And the ID of the commit which you can follow to get more info on what was committed



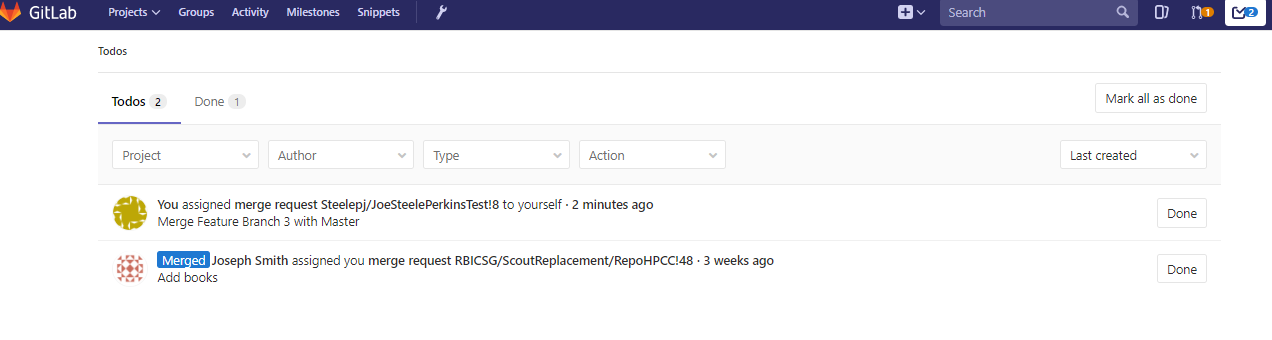
Then press “Compare branches and continue”

Here I can give the merge a name and assign it to someone. Here I have assigned it to myself.

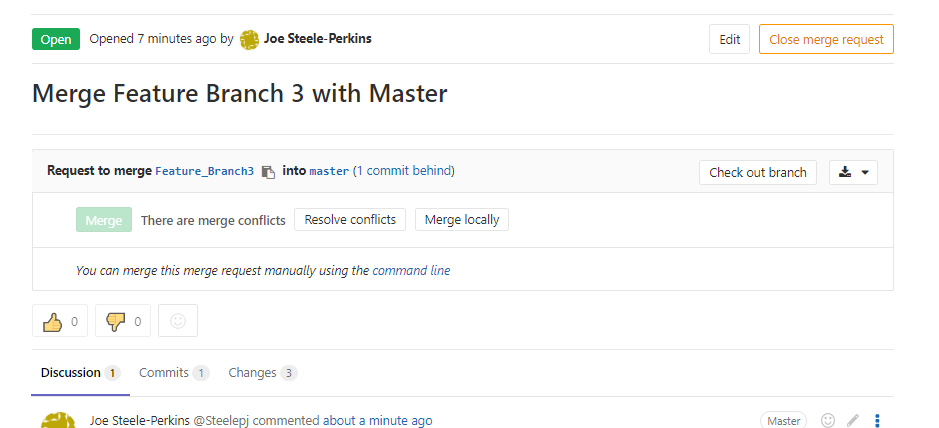


Then I submit the merge request.

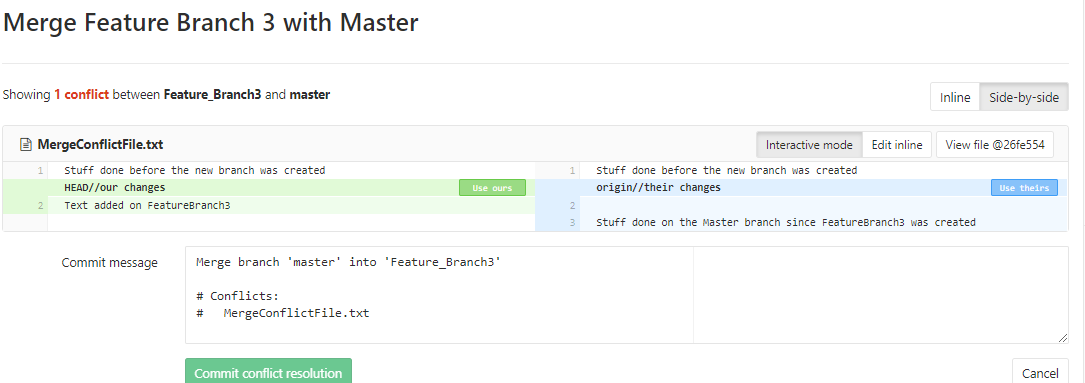
This merge appears in my ToDo list.



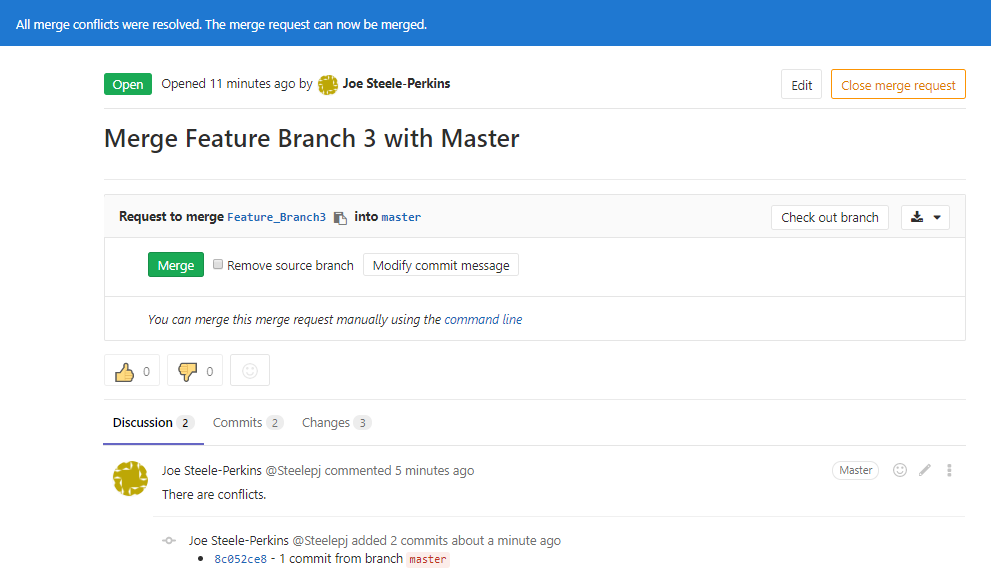
If I go into the merge it tells me I have conflicts.



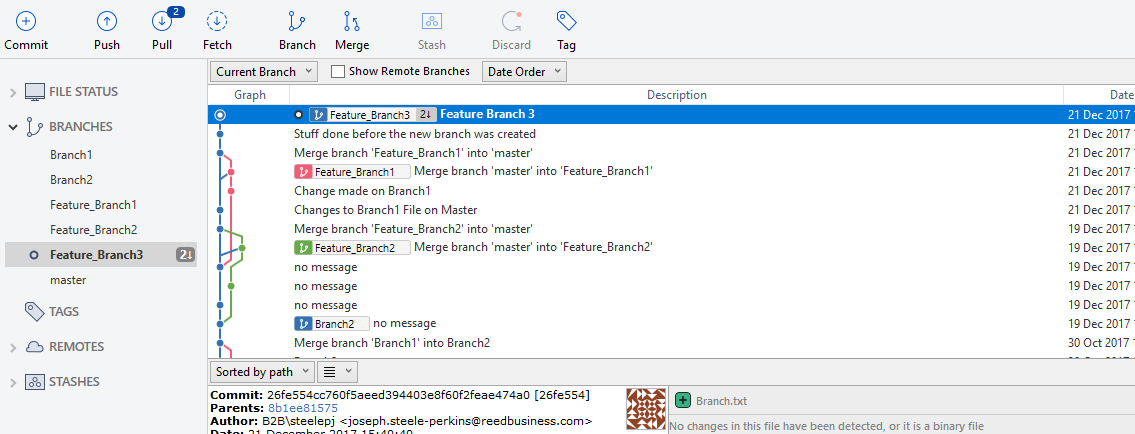
Let press Resolve conflicts…

“

I want to keep what is in Feature\_Branch3 so will select Use Ours. Then I press “Commit conflict resolution”. I get this page…

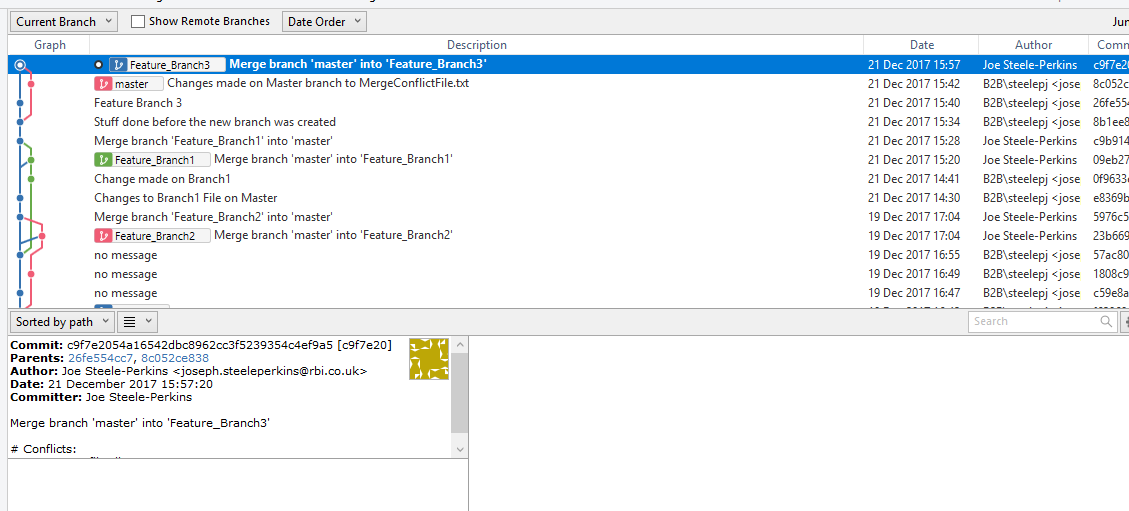


And look what has appeared in SourceTree. A commit has appeared against Feature\_Branch3. Why, when I haven’t committed anything to this branch.



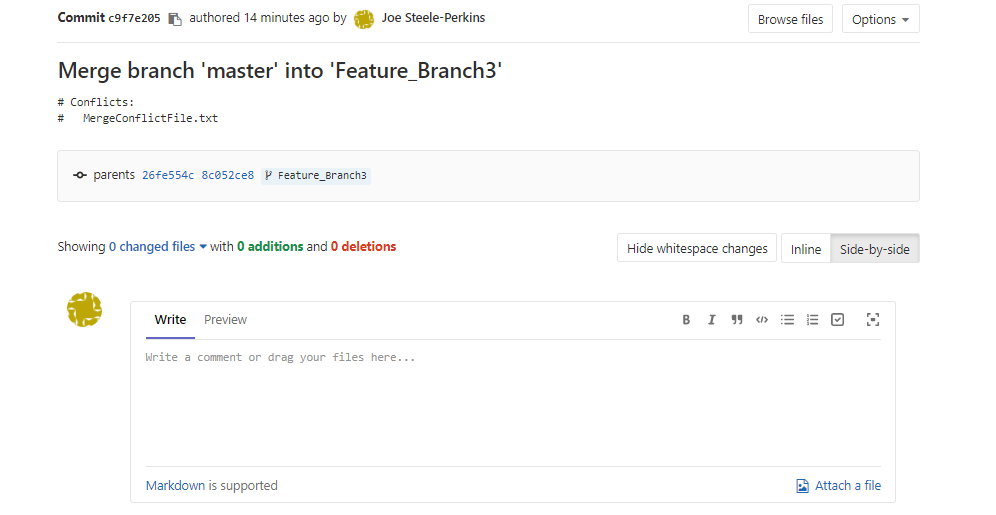
Let’s Pull to see what the commit was.

It is called “Merge branch ‘master’ into ‘Feature\_Branch3’”. Which makes absolutely no sense… This seems to have occurred because I resolved the conflict.



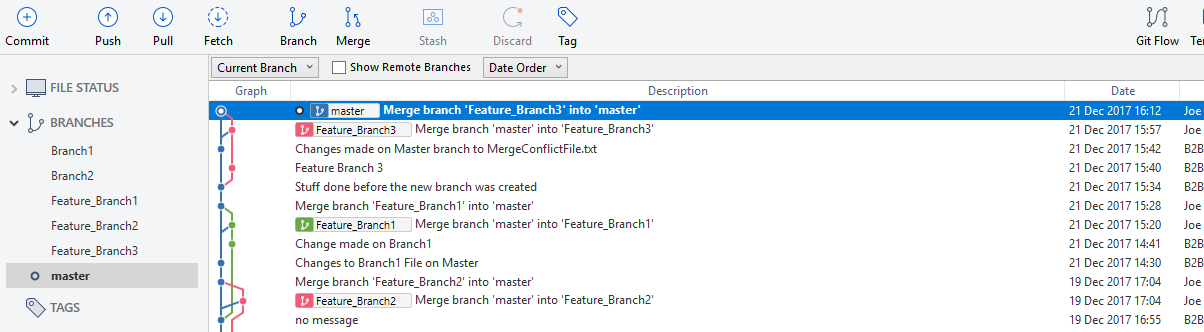
But the actual merge has not taken place. And this “commit” doesn’t seem to have actually done anything.

I can see the same commit in GitLab and again nothing has happened…



Okay, so let us actually merge in GitLab…

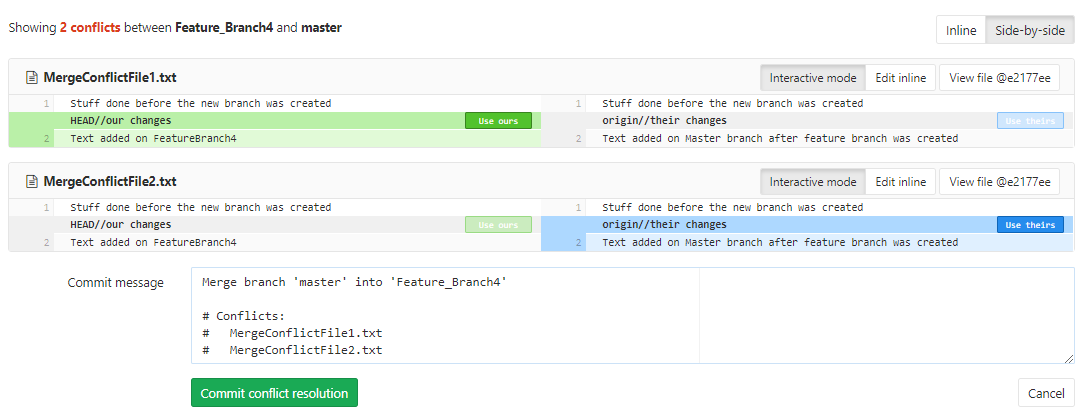
And now a correct commit has actually been created…



**Test 2**

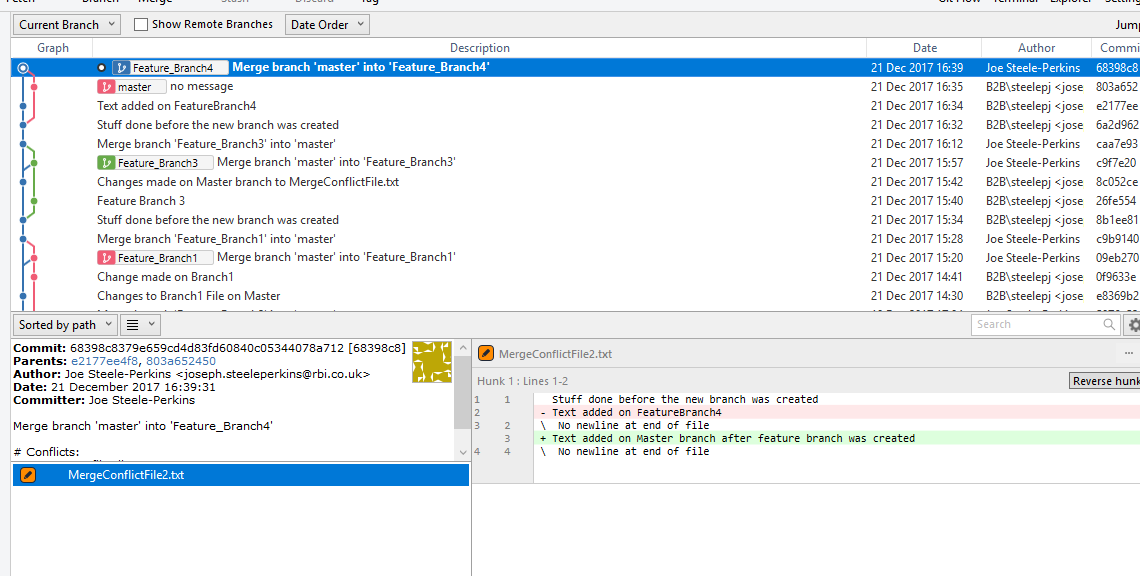
As above, but this time, let’s make it interesting and keep one Master conflict and one Feature branch conflict.

Okay, done all the way to Conflict resolution page…

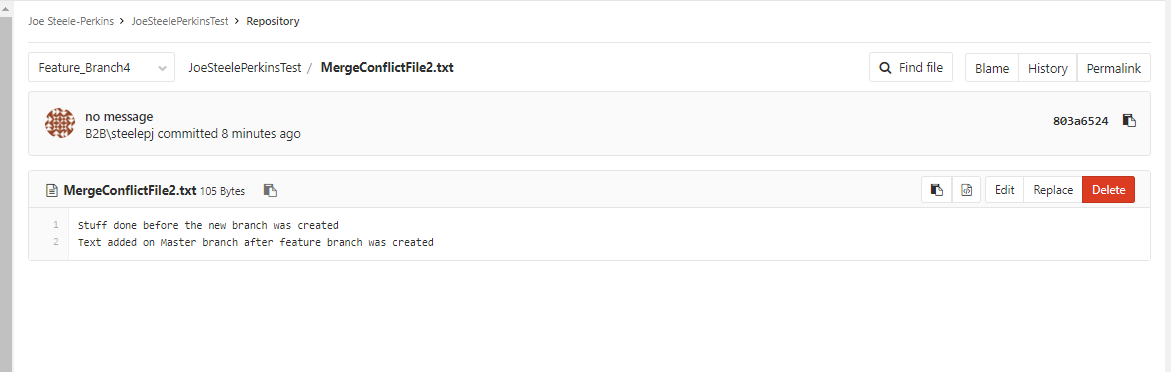


Decide to keep FeatureBranch4 version of MergeConflictFile1 and Master version of MergeConflictFile2.

Pressed Commit conflict resolution. Again get a misleading “commit” against FeatureBranch4. But this time there is a change…

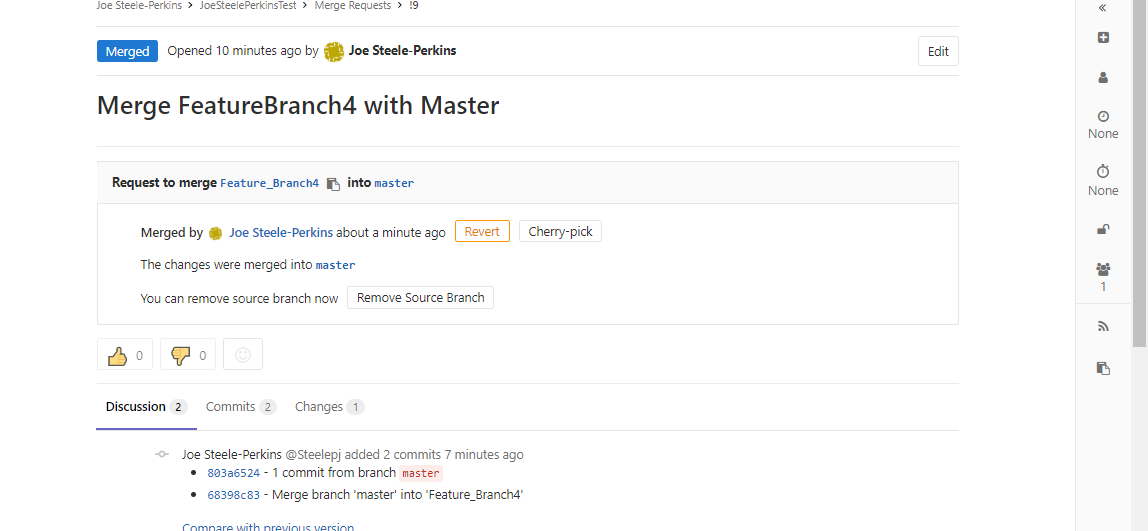


This time it is saying that it has overwritten the MergeConflictFile2 file with what is in Master. Let us verify if that has actually happened. Yes…



So, when you resolve conflicts, it is updating the source branch which fundamentally doesn’t feel correct. Surely the source branch should be unaffected by the merge?

Anyway, let’s continue with the merge.



And seems to have worked correctly on the Master branch and has rebased the feature branch. This is probably want we want. So the Developer is getting the feature branch into a state where it can be successfully merged with master.

**Roll-back a merge request**

Test 1 – simple, roll-back last merge.

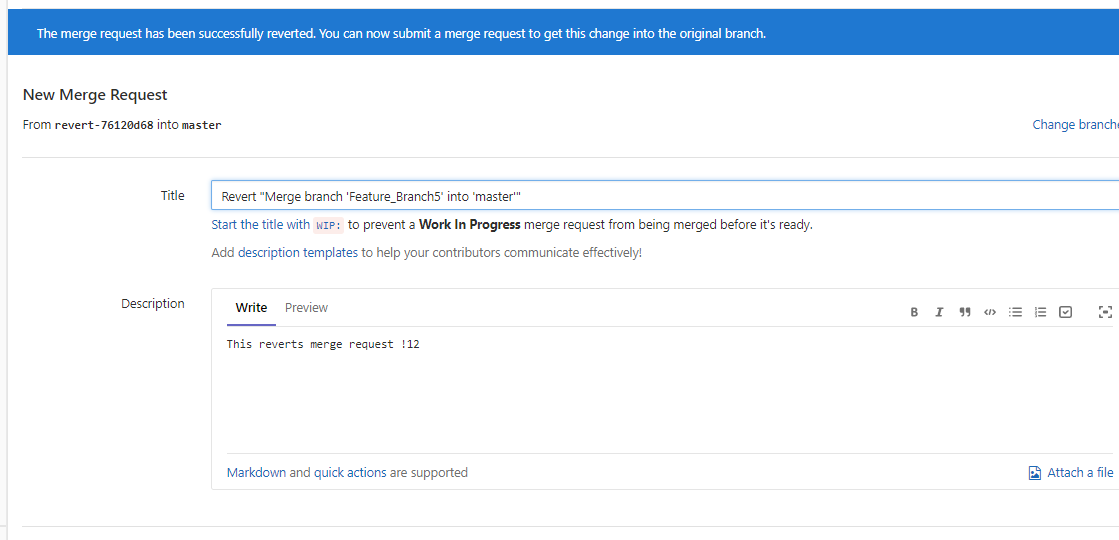
In Feature Branch 5, change MergeConflictFile3.txt and add a new file

In Master branch, change the MergeConflictFile3.txt.

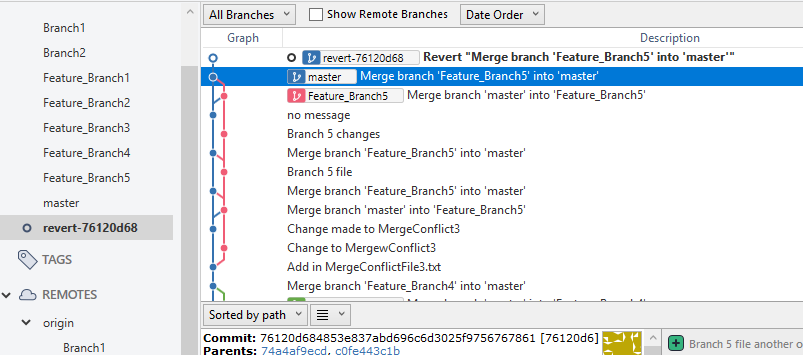
Merge Feature Branch 5 into Master branch. Resolve conflicts by keeping what is Feature Branch 5.

Go into Gitlab and reverse this merge.

First it does this…



What this has done is create a new branch…



And this contains the state of the file before the merge happened. But nothing has been changed in the Master or feature branch.

You then create a new merge request from the ‘revert’ branch to Master.

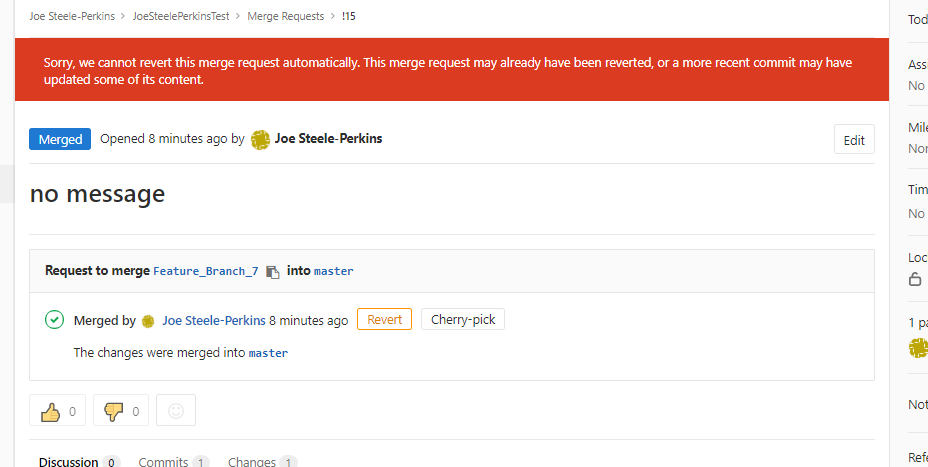
Test 2 – complex, roll-back second-to-last merge

Create new MergeConflictFile4.txt file in master and new feature branch “Feature Branch 6”

Update MergeConflictFile4.txt file and merge into master.

Then update it again and again merge into master.

Now try to reverse the first commit. You get this error message…

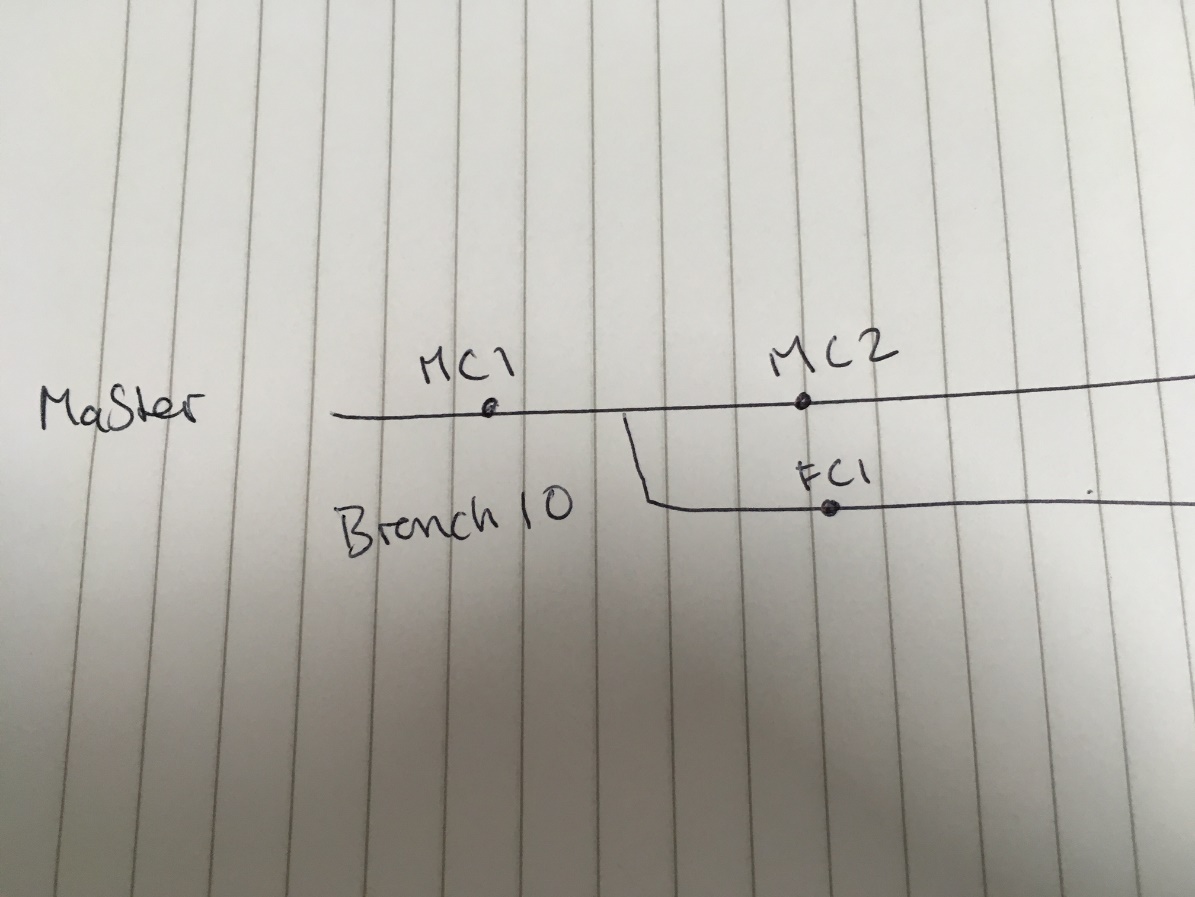


Is this possible if the two commits are on completely separate files?

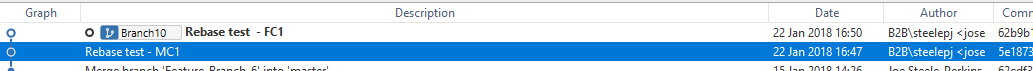
Yes

**Rebasing verses merging**

Created this structure



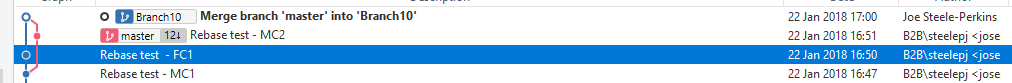
The commits on the feature branch (Branch10) look like this…



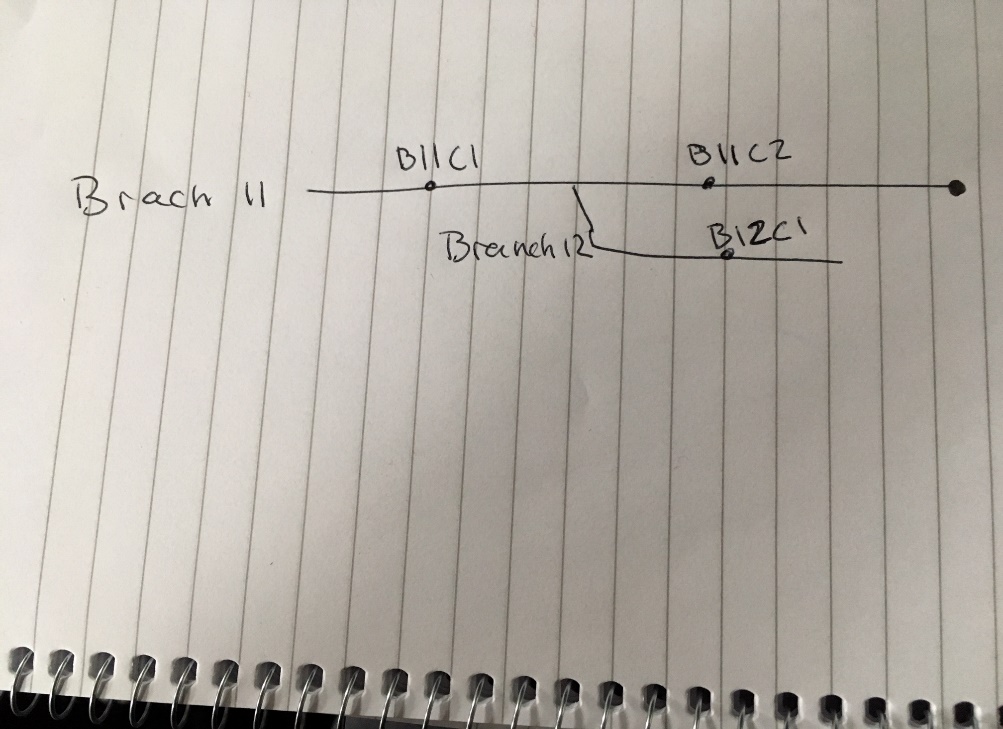
The commits on the Master branch look like this…



First try merging Master branch into feature branch. History now looks like this…

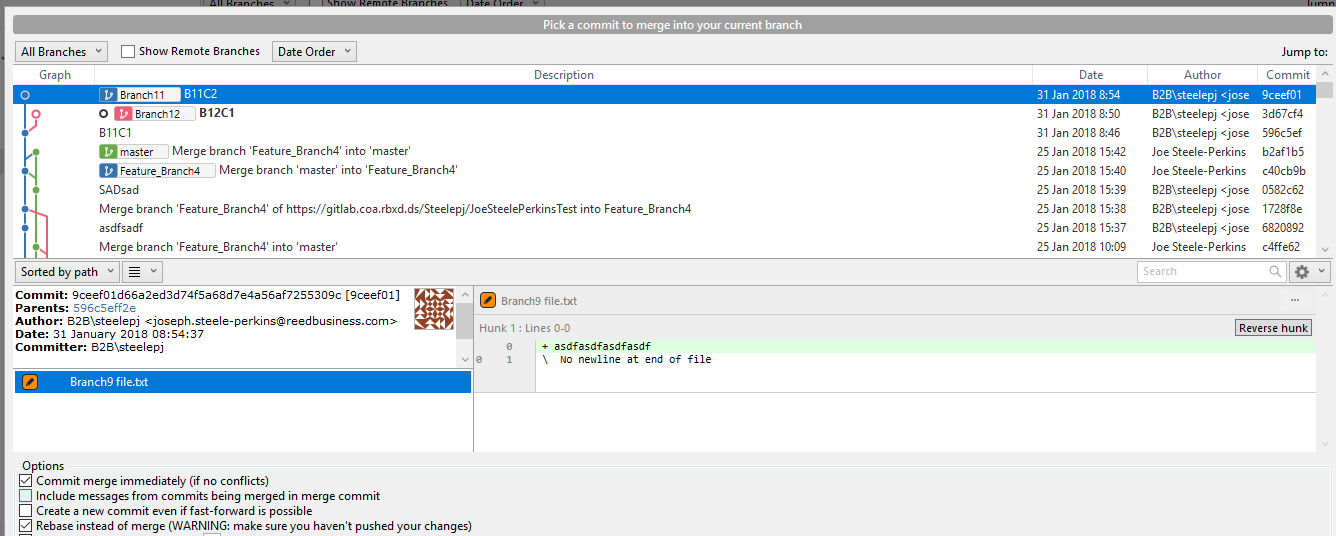


Next create this structure

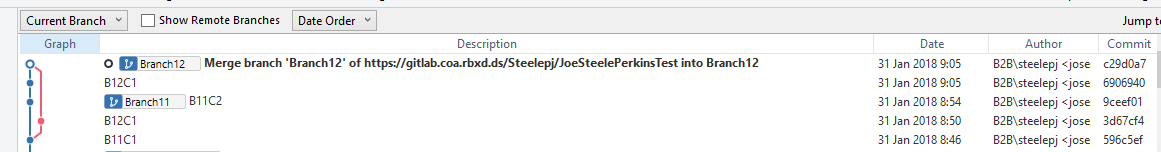


Now I want to rebase Branch12 from Branch11.

I do this in SourceTree. I go to merge branch 11 into branch 12 but select “Rebase instead of merge”



This is the result



All the commits are now on the branch but this final commit seems to be a bit meaningless.