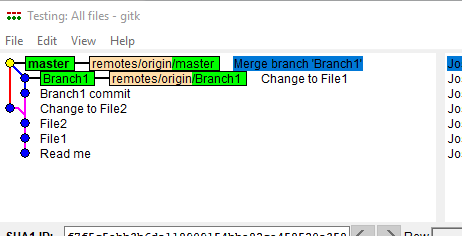
**How are Git branches and merges represented in log**

What log looks like after a merge

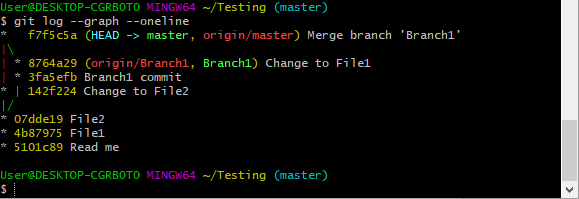
Gitk



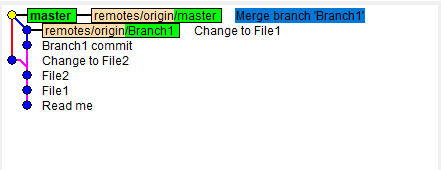
This shows two branches; master and Branch1.

It shows the point at which they split ie after “File2” commit. It shows two commits that occurred on Branch1 after the split; “Branch1 commit” and “Change to File1”. And one commit that occurred on the Master branch; Change to File2. Then it shows the merge back to the merge branch called “Merge branch ‘Branch1”.

And this is what it looks like in Git Bash



What happens if I delete Branch1? First do it locally.



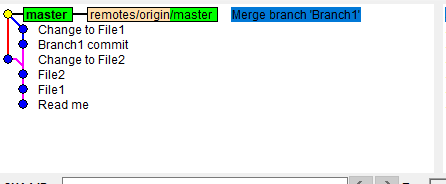
Structure is the same. There just now is no Branch1.

Now I’m going to delete the remote branch.

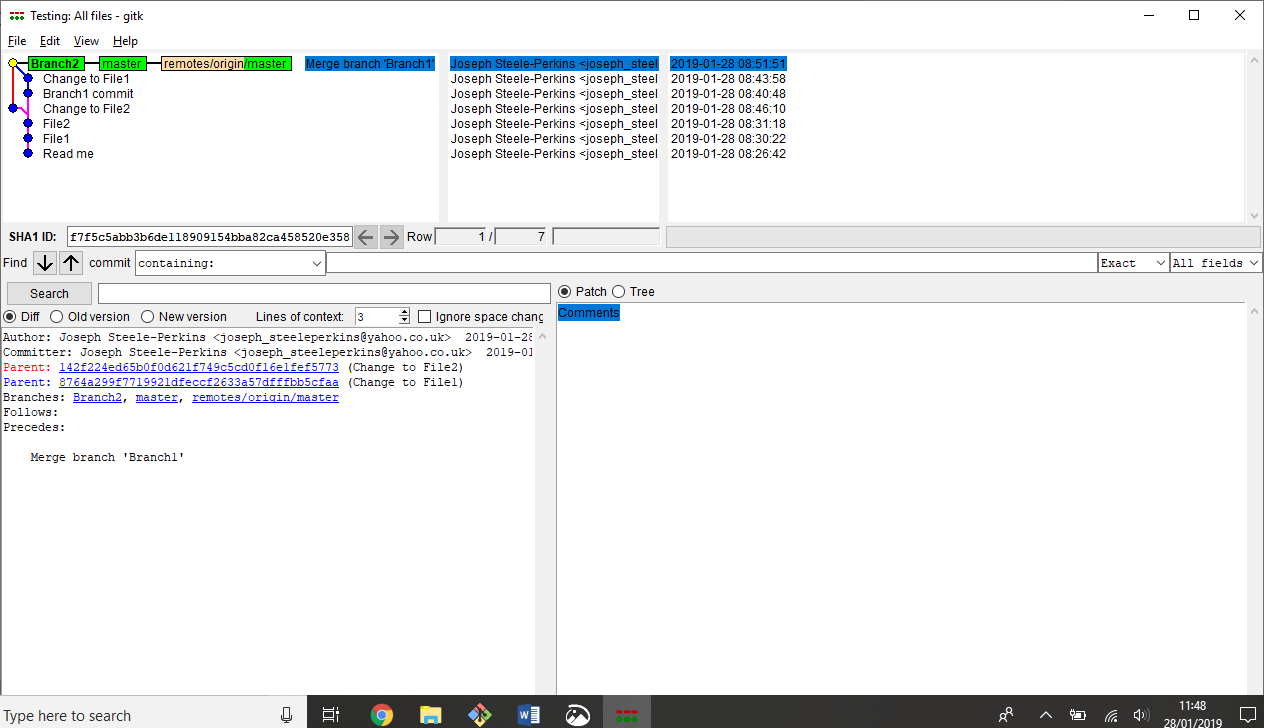
git push --delete origin Branch1

git fetch

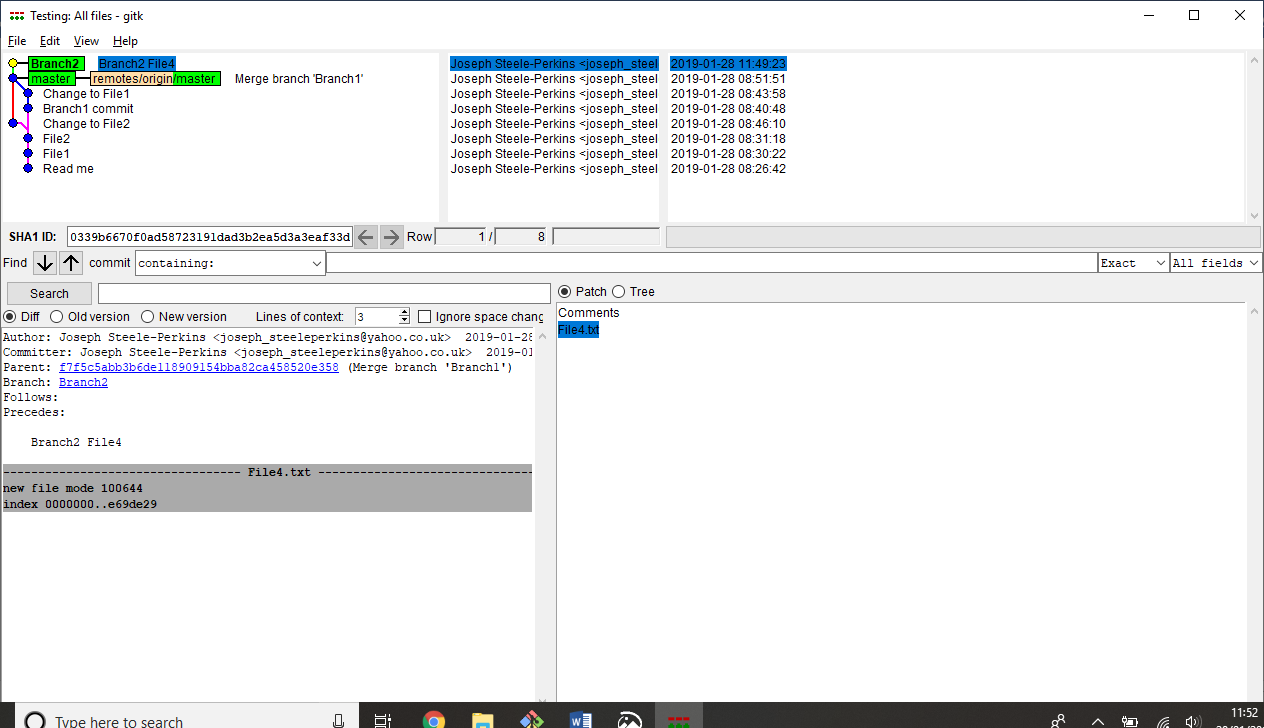
Structure is the same but no Branch1 label.



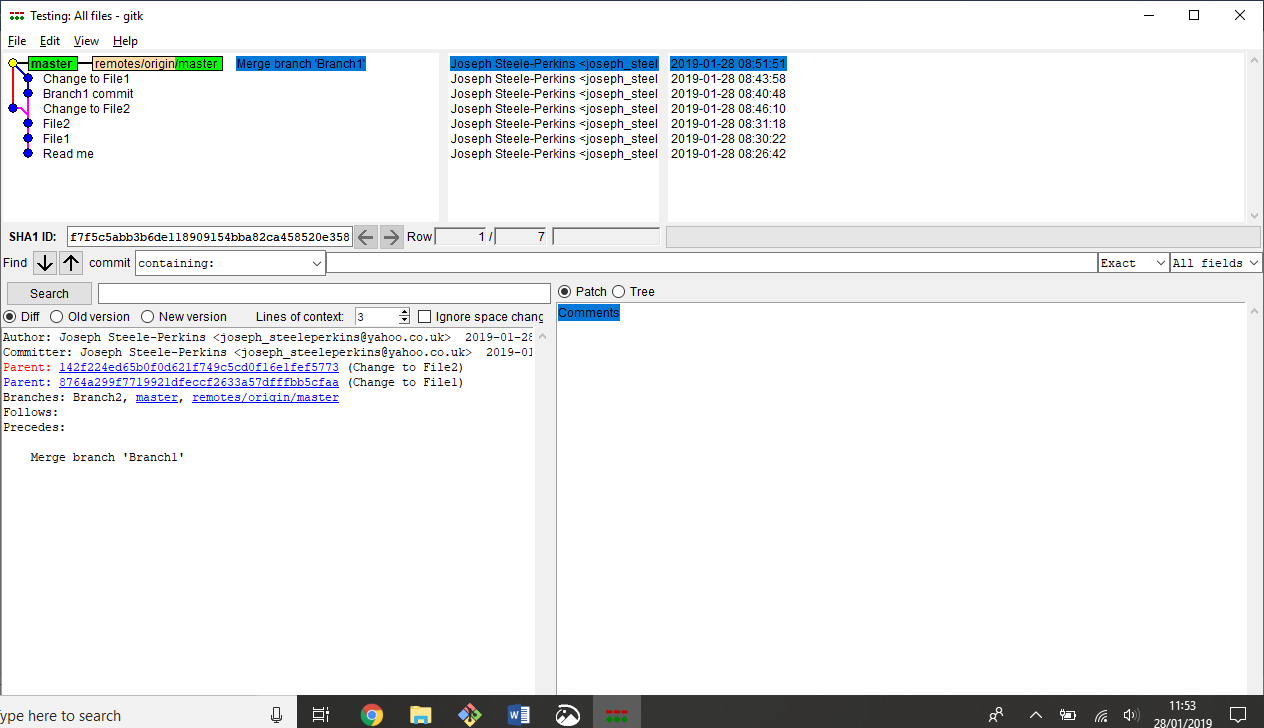
Create a new branch from master. No commits on it yet so is the same as above.



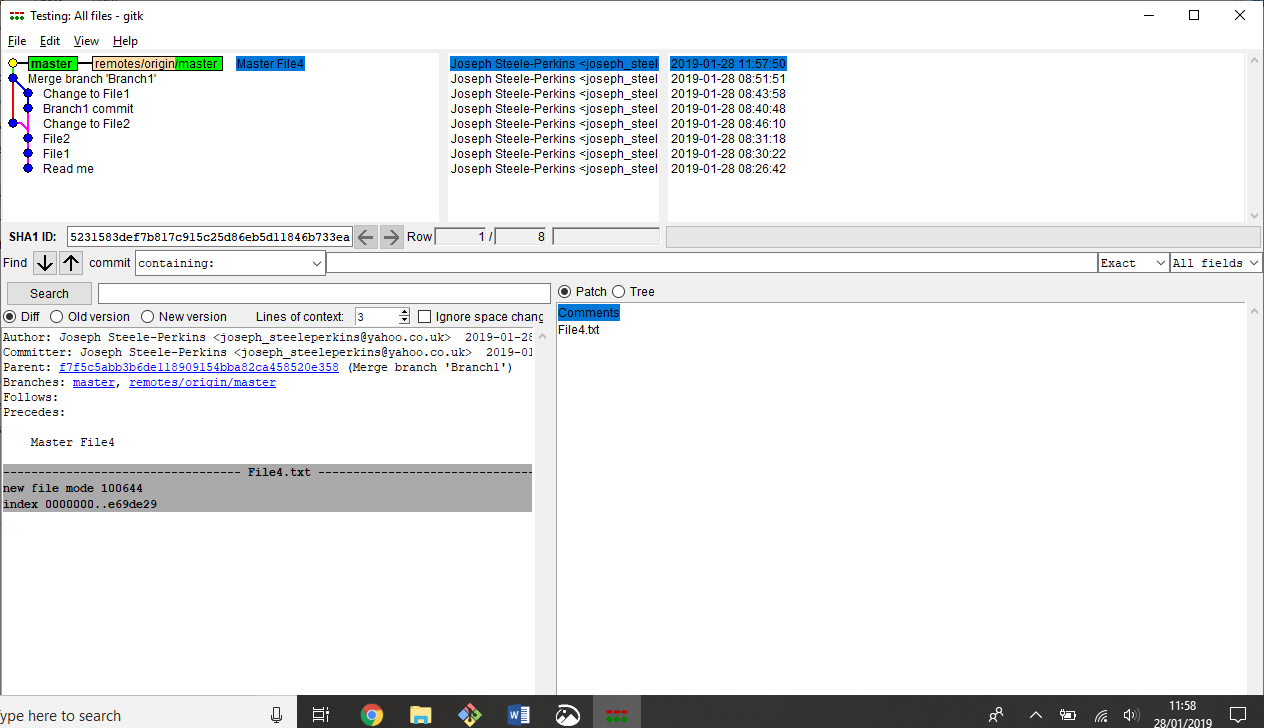
Create a commit on the new branch. It doesn’t look like a new branch yet.



Switch back to Master branch. It doesn’t know about Branch2 so hasn’t really changed

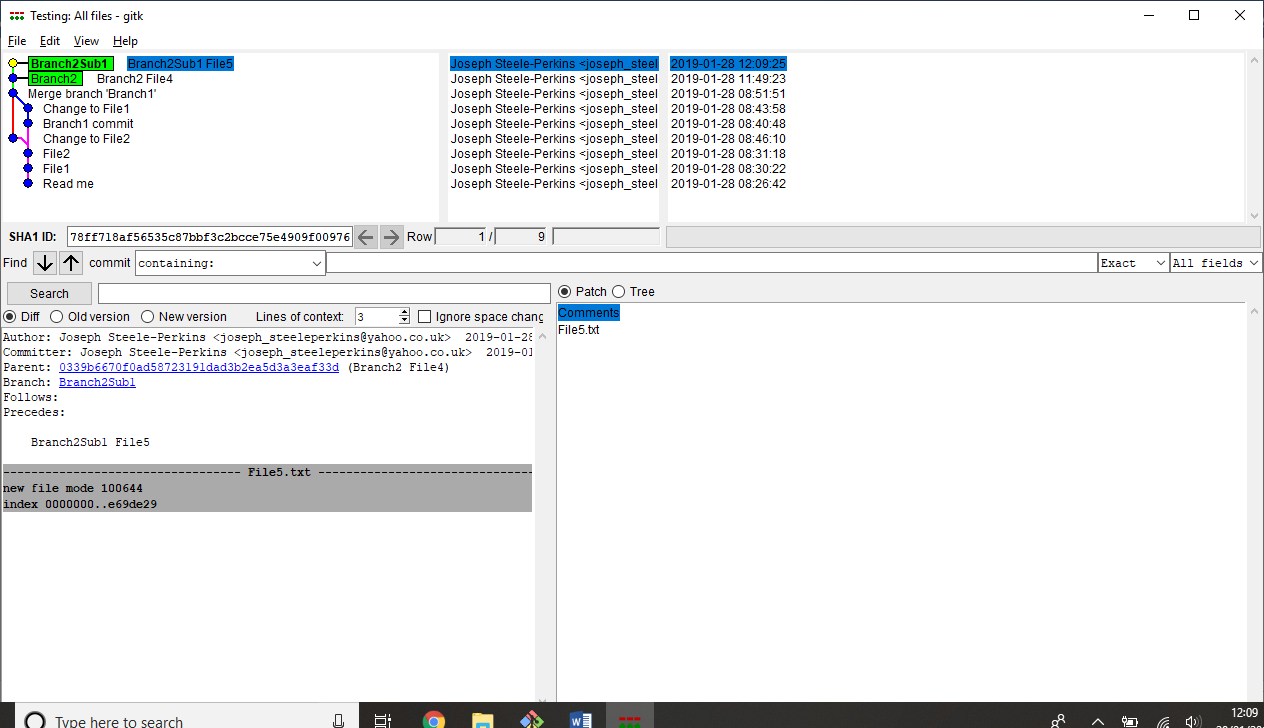


Create a commit on Master



Switch back to Branch2 and create a new branch Branch2Sub1 and create a commit on it.

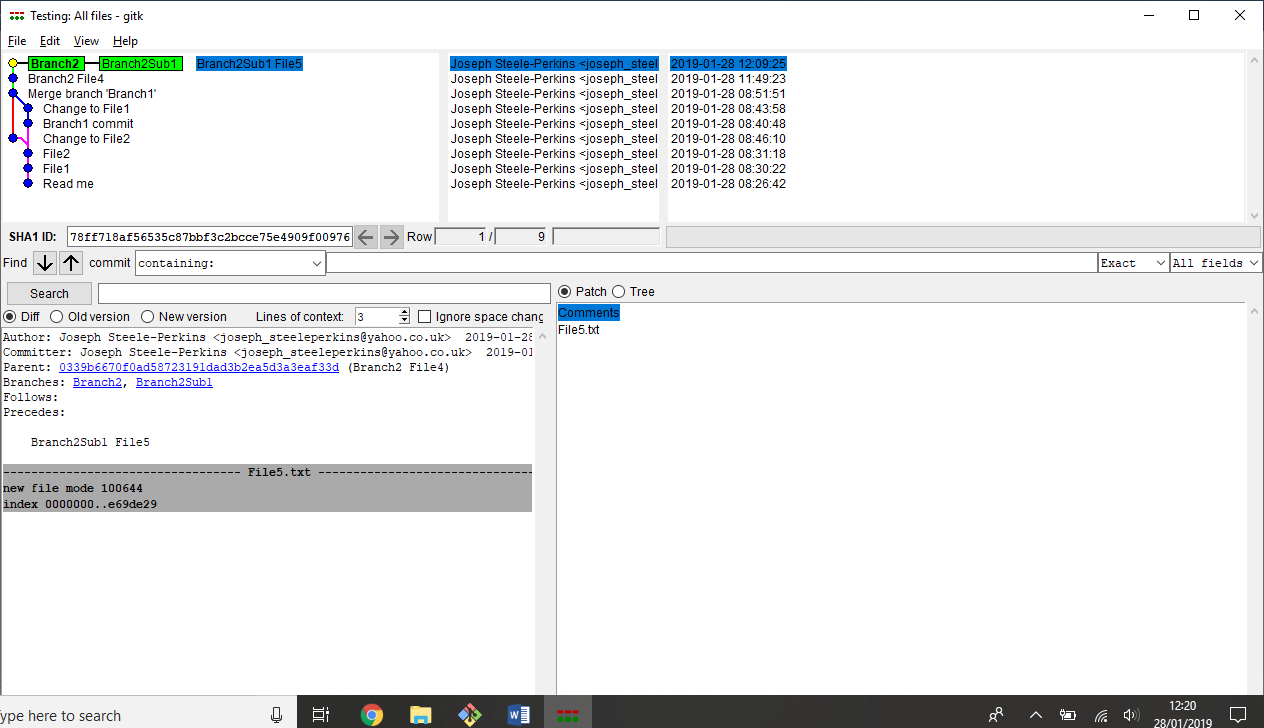
Note that nothing actually appears as a branch yet.



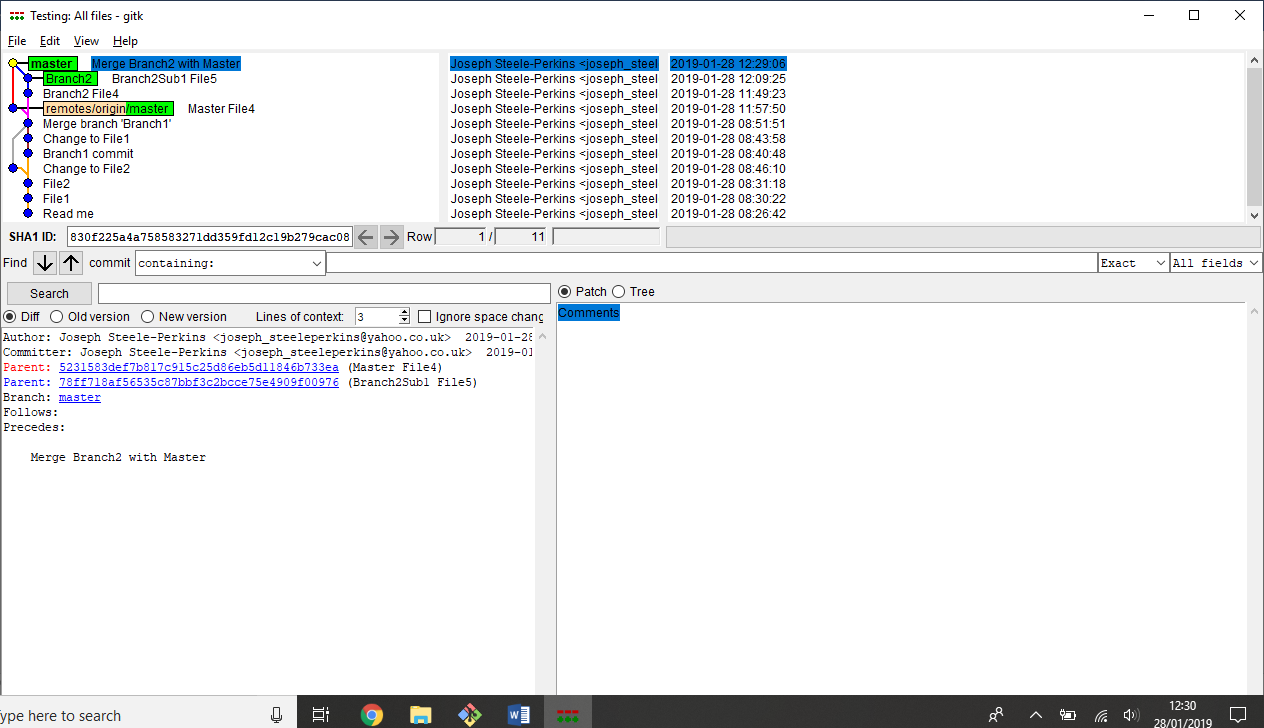
Lets merge Branch2Sub1 back into Branch2. Not that we not made any commits on Branch2 yet.

$ git merge Branch2Sub1 -m "Merge Branch2Sub1 into Branch2"

What is interesting is that the above merge doesn’t appear on the Branch2. And also Branch2Sub1 doesn’t show as a separate branch. I think it is because it didn’t actually do anything. It is not important in the history of commits. The merge commit will only be shown if the branches have diverged



Now I’m going to merge Branch2 back to master. This should show because the branches have diverged. And it does.

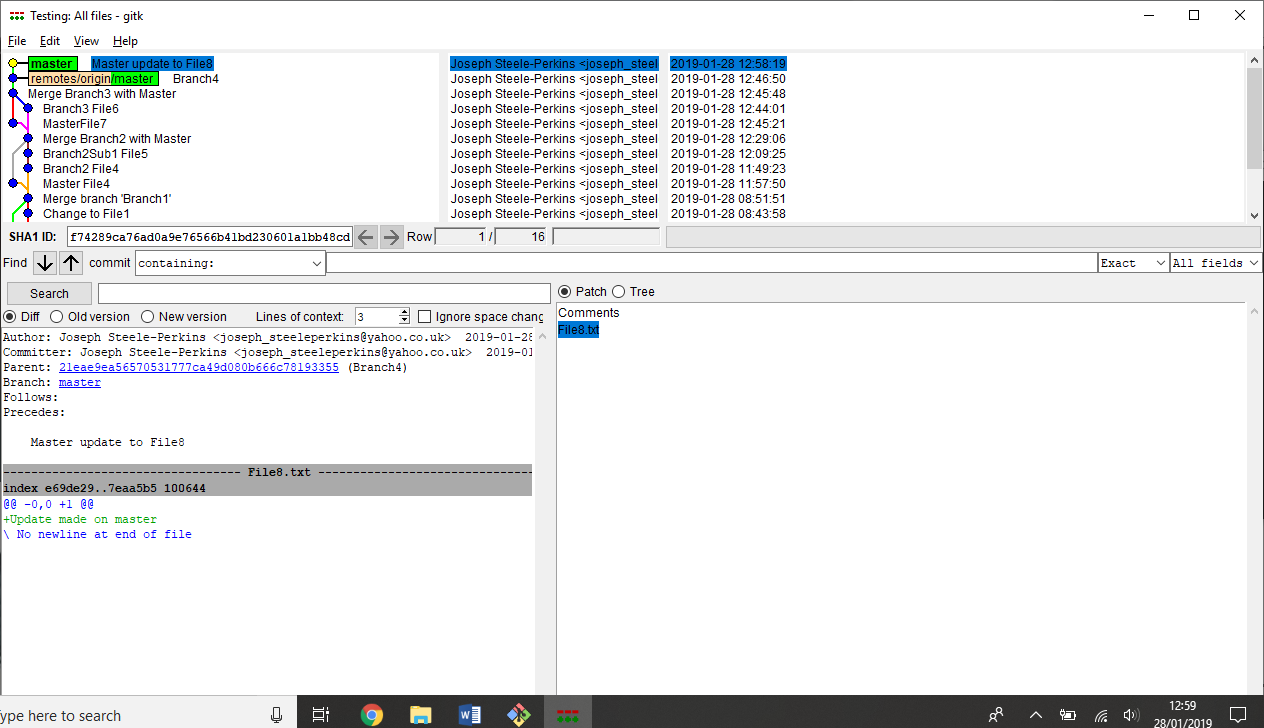


This merge commit hasn’t actually done anything though. Not really. There are no file changes. This doesn’t entirely make sense…

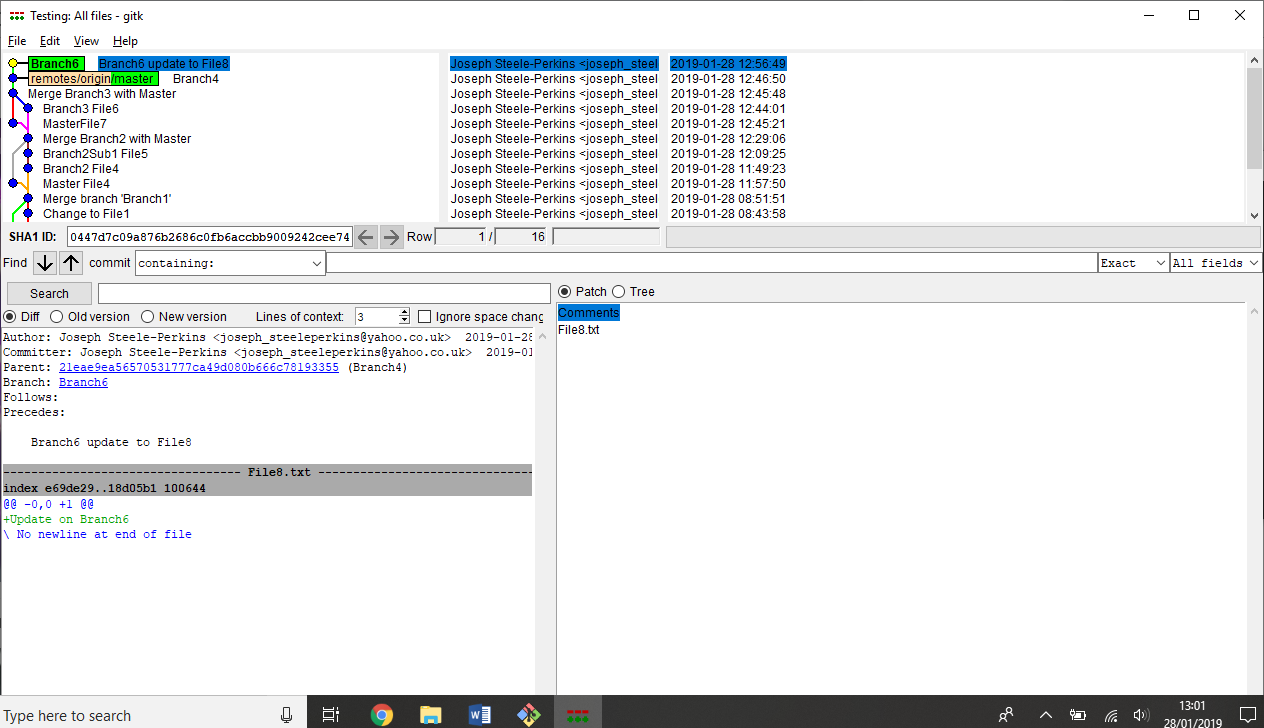
**How are merge with conflicts represented in log**

Create commit on master branch and Branch6 that updates the same file.

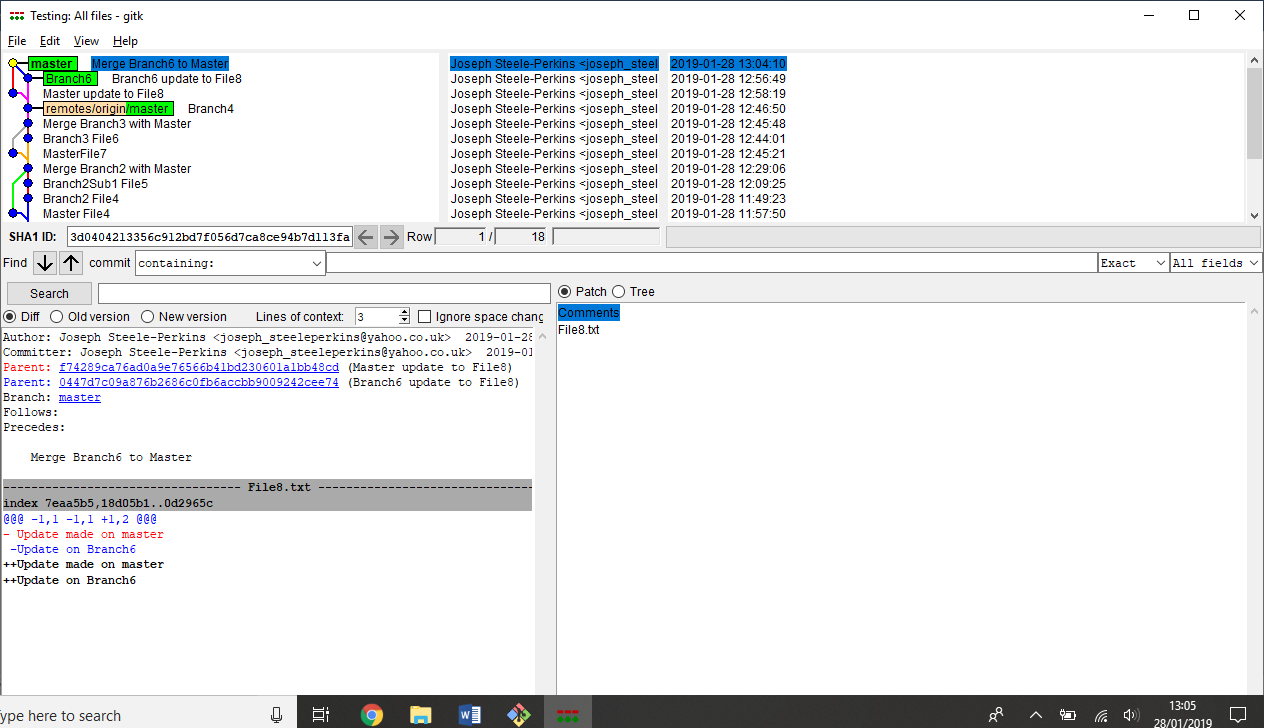
Master



Branch6



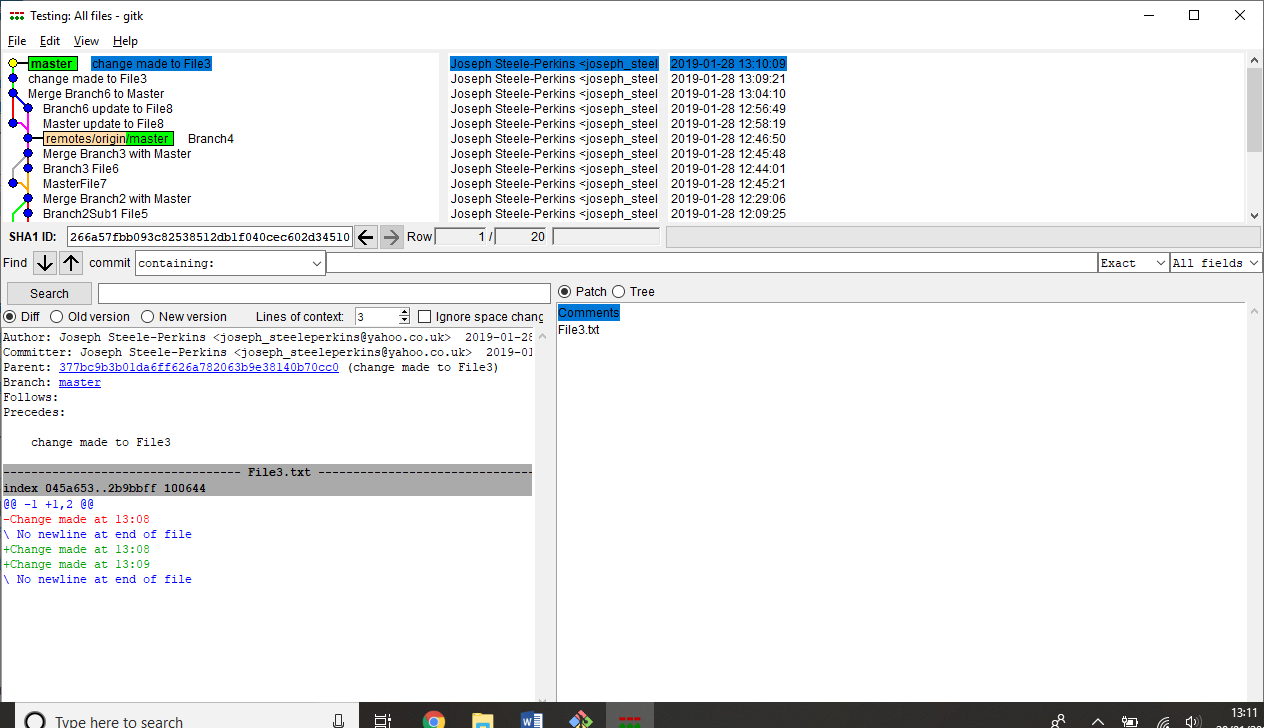
A commit that fixes conflicts looks no different that any other commit



**How to interpret Diff in GitK**

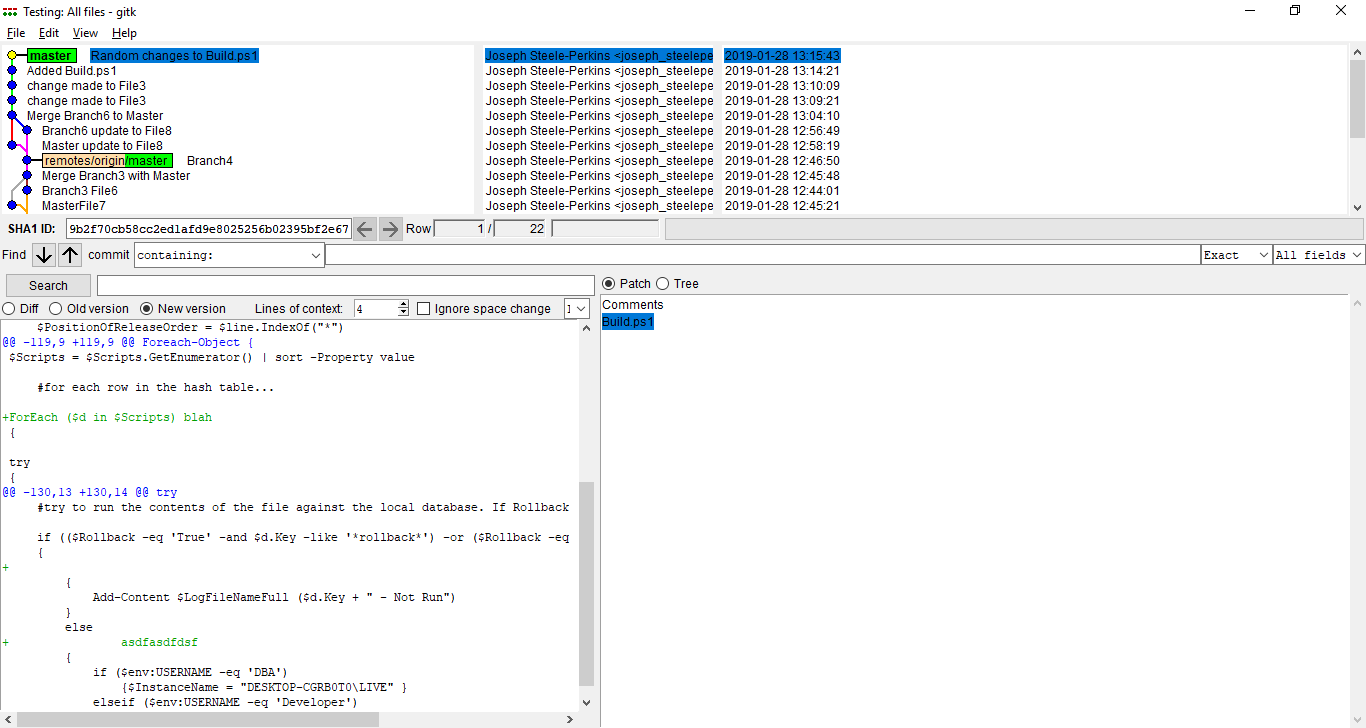
Made two commits on the master branch that changed File3

Looking at the below, red is what it was, green is what it is.



Lets try this with a real file.

Made some random changes to the Build.ps1. Removed lines, changed lines, added lines.

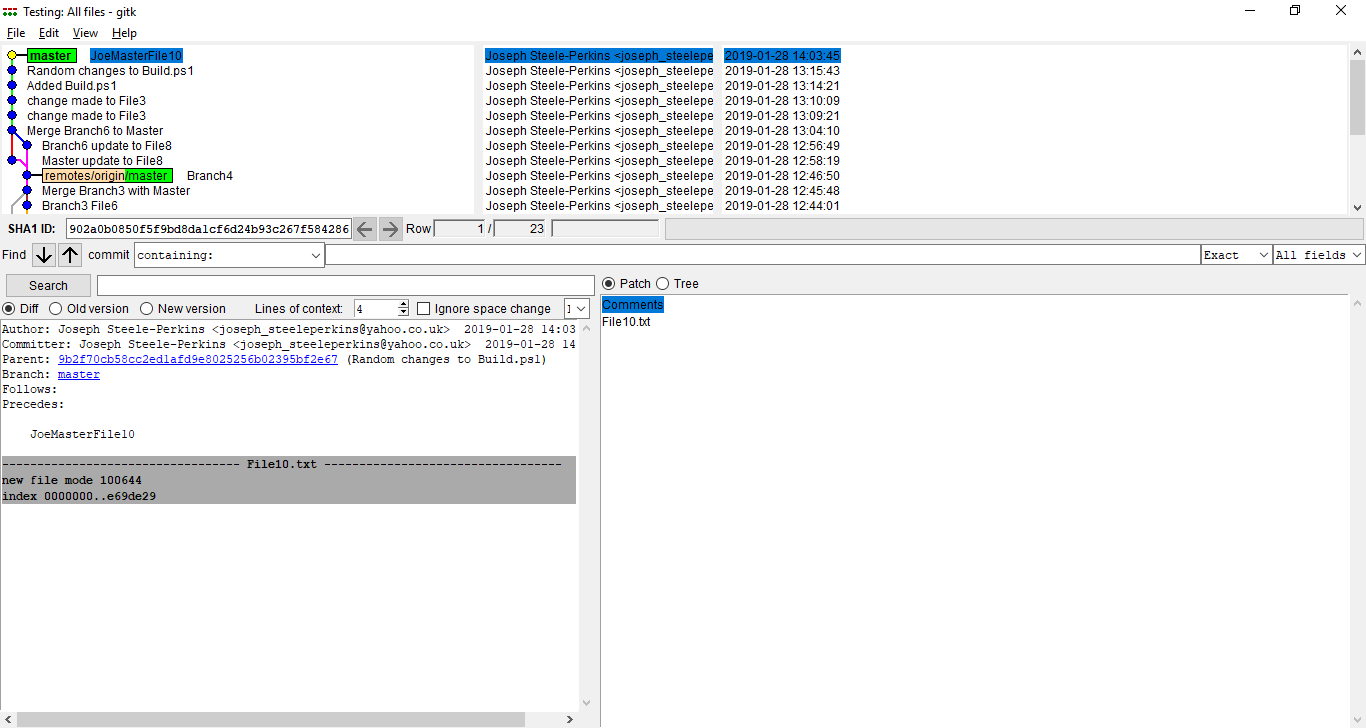


There are crosses next to everything that is new, including where it has been removed.

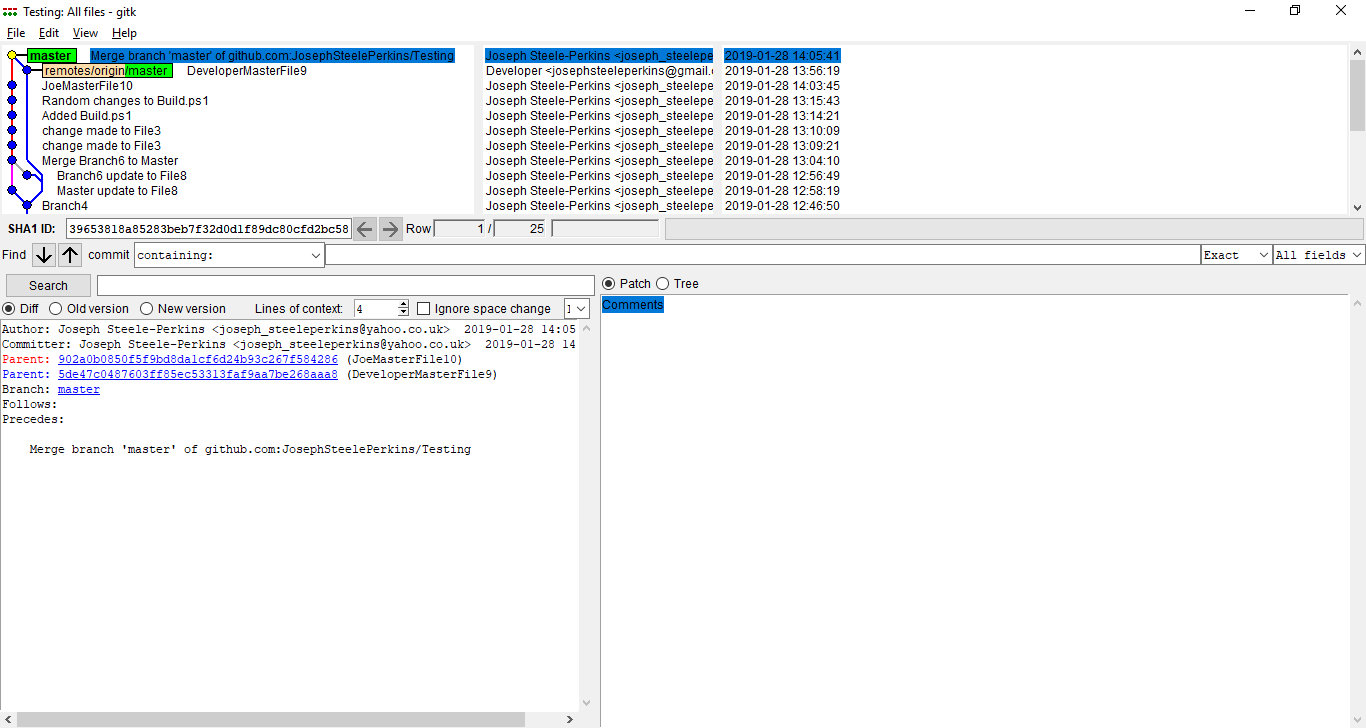
**How are pulls shown**

Me and another user have both committed to the master branch. These commits don’t conflict.

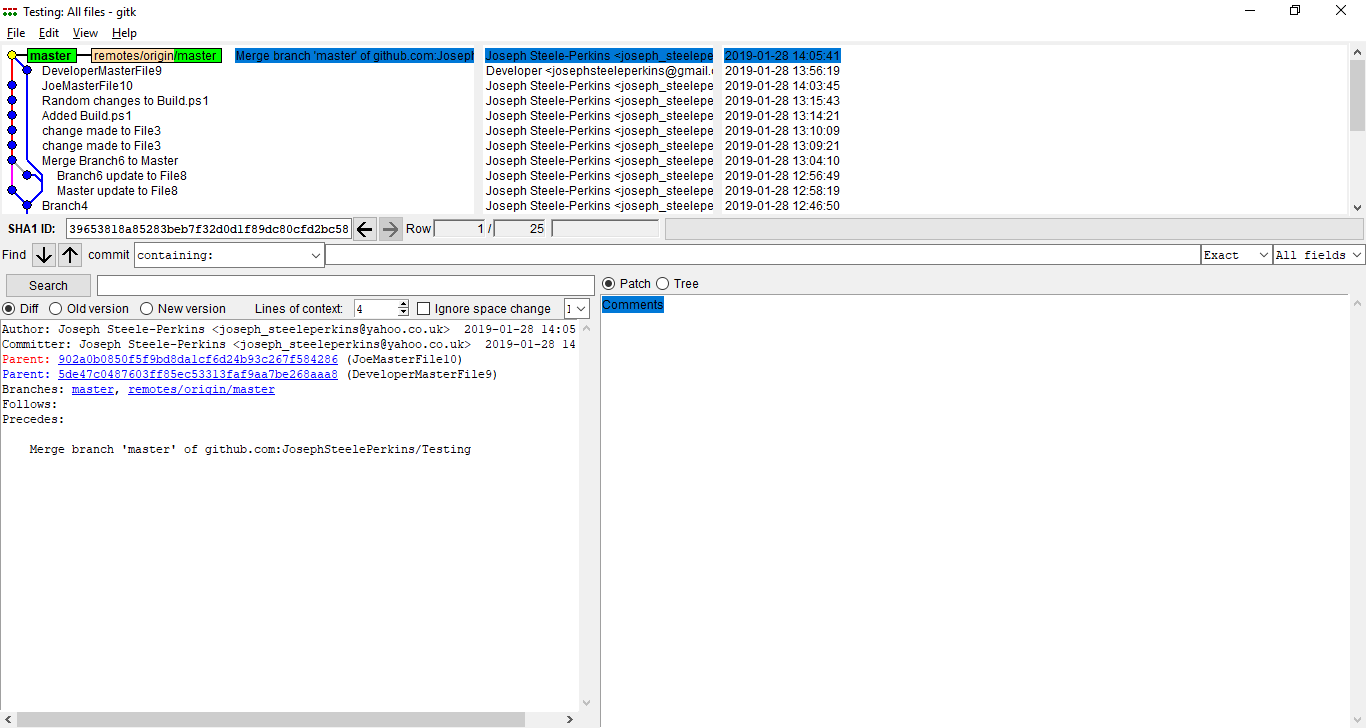
I have created the commit but not pushed it



Because the other use has a commit that I don’t I have to pull first. The pull is actually creating a merge. And this is what it looks like:

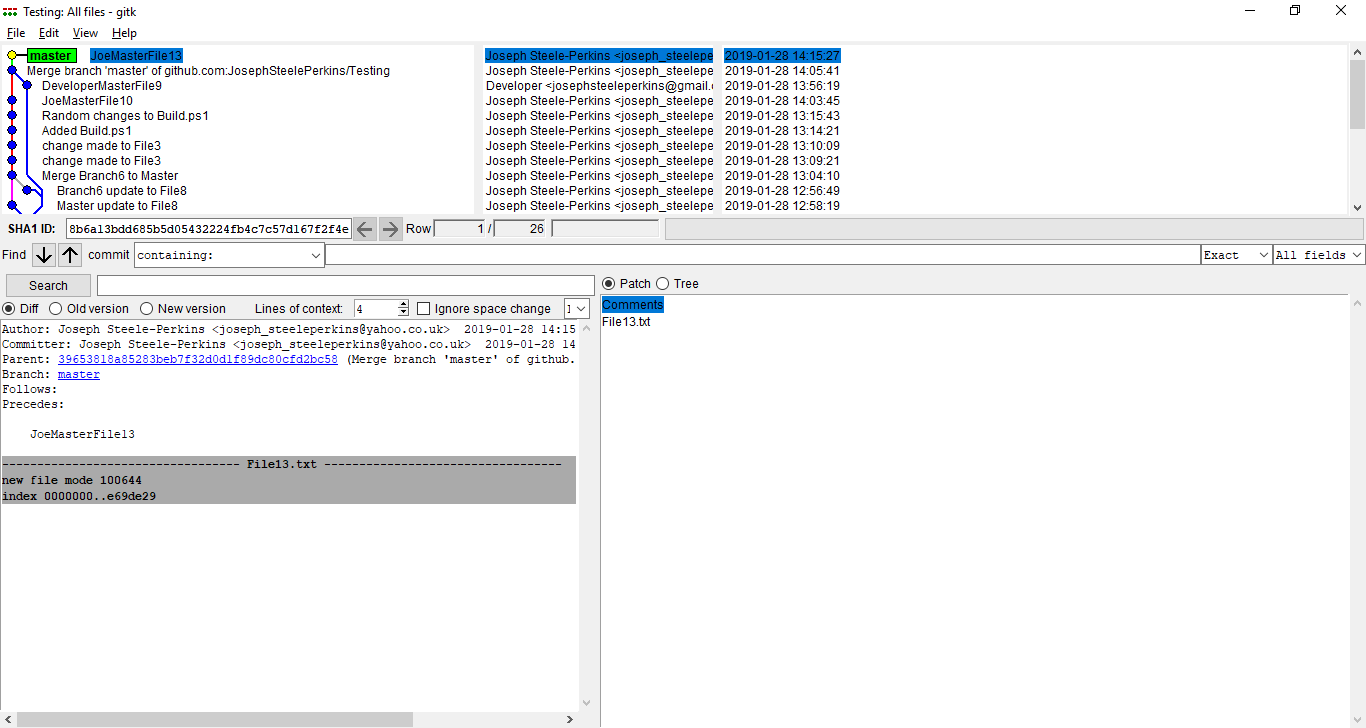


Because I did a merge, the other commit shows on a different ‘branch’. Now I will push.

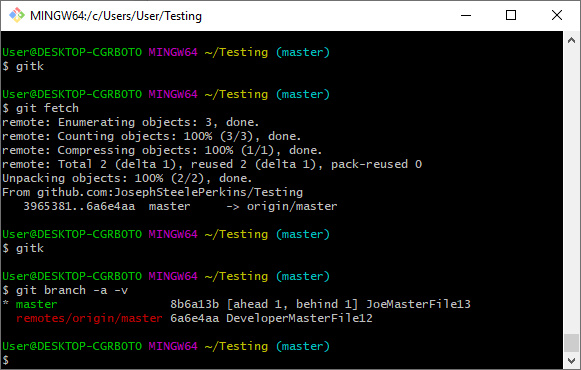


Here is a question; why is the parent of the DeveloperMasterFile9 commit “Branch4”? Is it because I didn’t push the master branch through all those commits? Lets do another example now I know the remote is up to date.

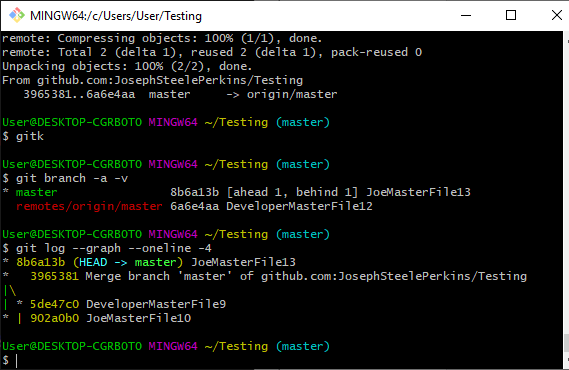
Reproduced right up to the push. I’m going to do a fetch to see what that looks like.



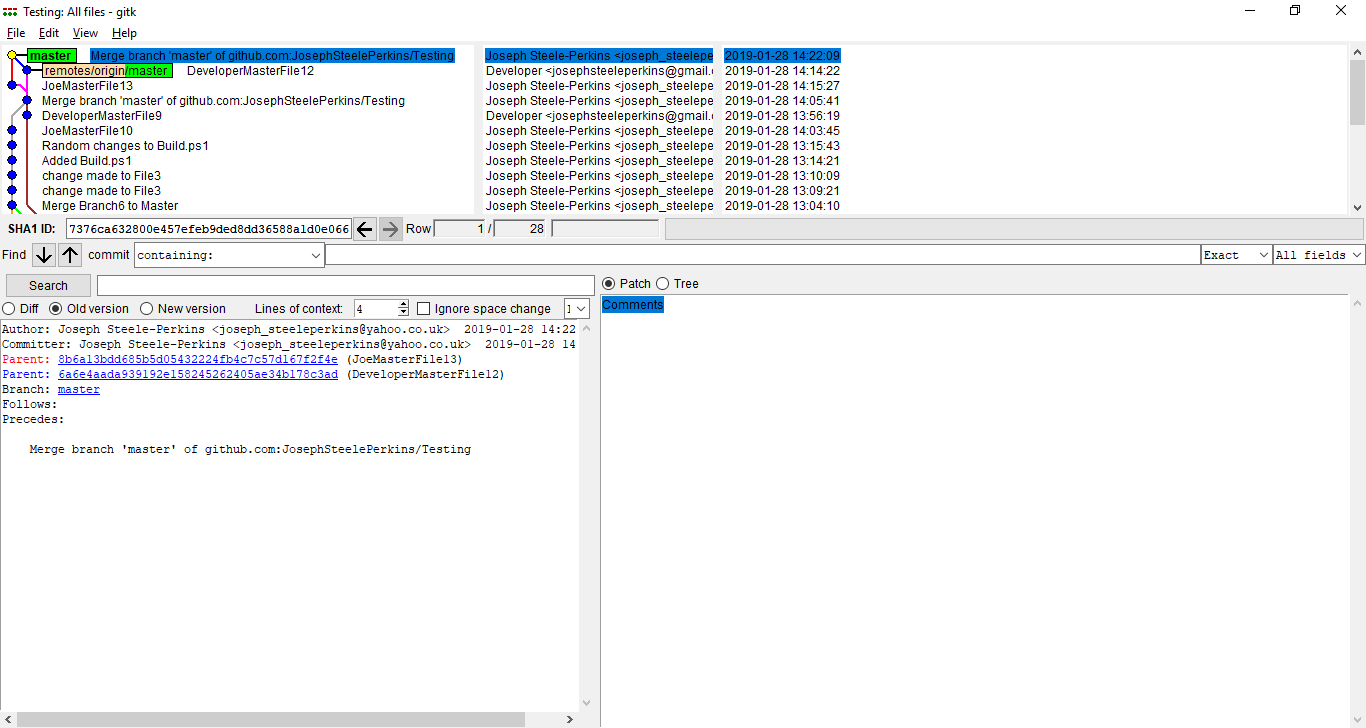
Interesting! Where is origin/master? Lets check what Git Branch can see.



It can see that it is on DeveloperMasterFile12 commit but GitK can’t. Git log can see it.



Now I’m going to do the pull. Now looks okay



The lines in GitK show parenthood of a commit. They are not necessarily branches. And branches will not necessarily show on GitK.