Guide to Get Good at Git

For the steps below, I used the name BillyBranch as my own branch name. This would be the main branch I write my code on. Obviously for you guys it would be EvertonBranch or OussamaBranch or whatever you want to name it.

**Prerequisites:**

* Download Git Bash terminal and connect it to your github. We did this in the HTML/CSS class and I don’t 100% remember it but it’s probably on stack overflow somewhere. You can test if you have it by either just pressing the windows key and typing ‘Git bash’ or right click on a folder in File Explorer and see if there’s an option to ‘Git Bash Here’.
* Set up visual studio to be your default merge tool. This will help you when you hit merge conflicts. The built-in merge tool is VIM but it’s fucked up and hard to use so I would recommend using Visual studio: <https://stackoverflow.com/a/59773720/12477301>

## Key Things to Know About Git:

Github is all about local and origin (also known as remote) files. It uses ‘master’ and ‘branches’. In HTML/CSS class, all we used was master. With this project, we will use ‘master’ as a ‘community file’, where our main project will be. We will each have branches where we write our code, and bit by bit, we will add our code to ‘master’.

You have a local master, and Github has an origin/master. Think of origin master as “everybody’s” master.

The general procedure is:

-Checkout **origin/master** as a local branch called **‘master’**

-Make a local branch off that called **BillyBranch**

-Make changes to **BillyBranch**

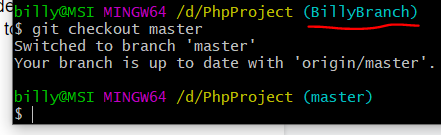
-Make a pull request for **BillyBranch**, which gets approved

-**origin/master** gets *merged* with **BillyBranch.** Whatever new code was in **BillyBranch** is now also in origin/master. **BillyBranch** is now deleted.

-Update your local **‘master’** to be inline with **origin/master**

-Repeat from Step 2

Before writing any code or making changes, always ensure you have your BillyBranch checked out: RED = GOOD



If you accidentally change code while having master checked out, it’s not a big deal, it is just tedious for you to fix. However**, never have master checked out and write**:

git push -u origin master

**Even though all we did in HTML/CSS was git push -u origin master, you should never do this.**

It will overwrite the **origin/master**, which, remember, is EVERYONEs master. Although, for us this isn’t a huge deal, because we can just move it back one commit. It’s not the end of the world at all, it may just require some file shuffling.

## Important Commands/Command Info:

**git fetch origin** → basically “updates” your git with all the github tracking info. So if a new branch was made by one of us, or one gets deleted, git fetch origin will now update your local git so it “knows” about this new branch.

Two commands that look the same:

1-**git checkout -b *branchName***

2-**git checkout *branchName***

Command 1 does the following commands in one go:

git branch BillyBranch //which creates a local branch called ‘BillyBranch’

git checkout BillyBranch // which puts you on that newly made local ‘BillyBranch’

Command 2 puts you on that branch, but only if it exists.

When you call ‘git branch *BranchName*’, as stated above, it creates a local branch called ‘*BranchName*’. It is important to know that when you do this, it **creates a copy of whatever branch you’re currently on.**



This will create a new branch called MyNewBranch, and it will copy whatever **master** is.

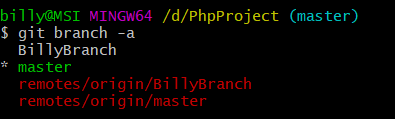


This will create a new branch called MyNewBranch, and it will copy whatever **BillyBranch** is.

**git branch -D *branchName*** //Deletes a branch

**git branch -a** //Show all local branches

For me, this would show:



BillyBranch: the branch you will be coding on.  
\* Master: Your local version of the origin master.

-The Green and \* mean it is your current branch

Both the remotes are the github version of your two branches.

Once other people make their branches, they will appear under the remotes too.

-Note you will probably have to call ‘**git fetch origin**’ to update your git tracking in order to see them.

With your local master checked out:

**git pull origin master**

Will make **your local master up to date with origin/master**. This will be important later.

## Pulling the Repo, Making Your Branch, Committing Changes:

1. Go to wherever you would like to make your Git Repo. For this example, I made a new folder PhpProject in my D drive: D:/PhpProject. When we actually make this, we will probably have to put it in the correct ampps/www folder.
2. On your newly made folder, right click and select Git Bash Here.
3. In git bash, write ‘git init’
4. In git bash, write ‘git clone <https://github.com/cook0318/PhpProject.git>’
5. In git bash, write ‘git remote add origin <https://github.com/cook0318/PhpProject.git>’
6. At this point, you will automatically have your local ‘master’ checked out.
7. In bash, write ‘git checkout -b *BranchName’* to create your new branch.
8. From here on out, use the following commands:

git add -A //adds all files

git commit -m “Commit Message.” //commits w/ message

git push -u origin *BranchName*

This last command will not only push your changes to your local branch, but it will **create a remote branch**, so that anyone can see it.

A few common situations:

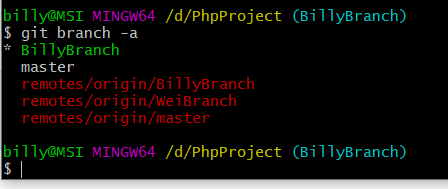
### **I want to checkout someone else’s branch.**

I am currently on BillyBranch

I want to check out Wei’s branch. I run

git fetch origin // to ‘update’ my laptop so it knows all the other gits in this repo

git branch -a // to show all branches

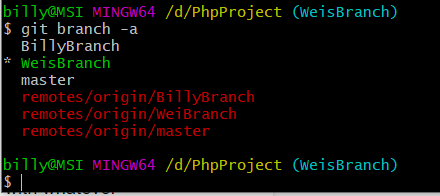


In order to ‘checkout’ Wei’s Branch, you have to *make your own, local ‘WeiBranch’*. It will be up to date with whatever **remotes/origin/WeiBranch** is. You run:

git checkout -b **WeisBranch** origin/WeiBranch

where **WeisBranch** is going to be the **name of your local branch.**

Running git branch -a shows us:



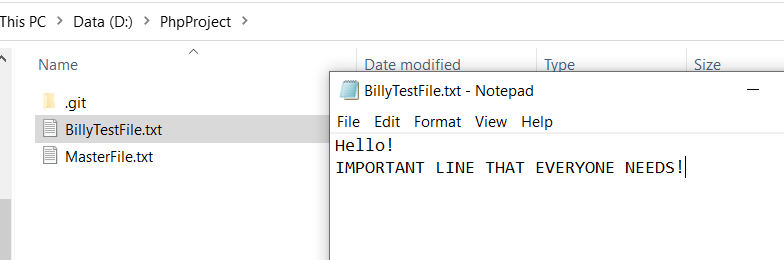
Now, I am on my local version of WeisBranch.

### **I am happy with my changes and want to push them to master so others can use them.**

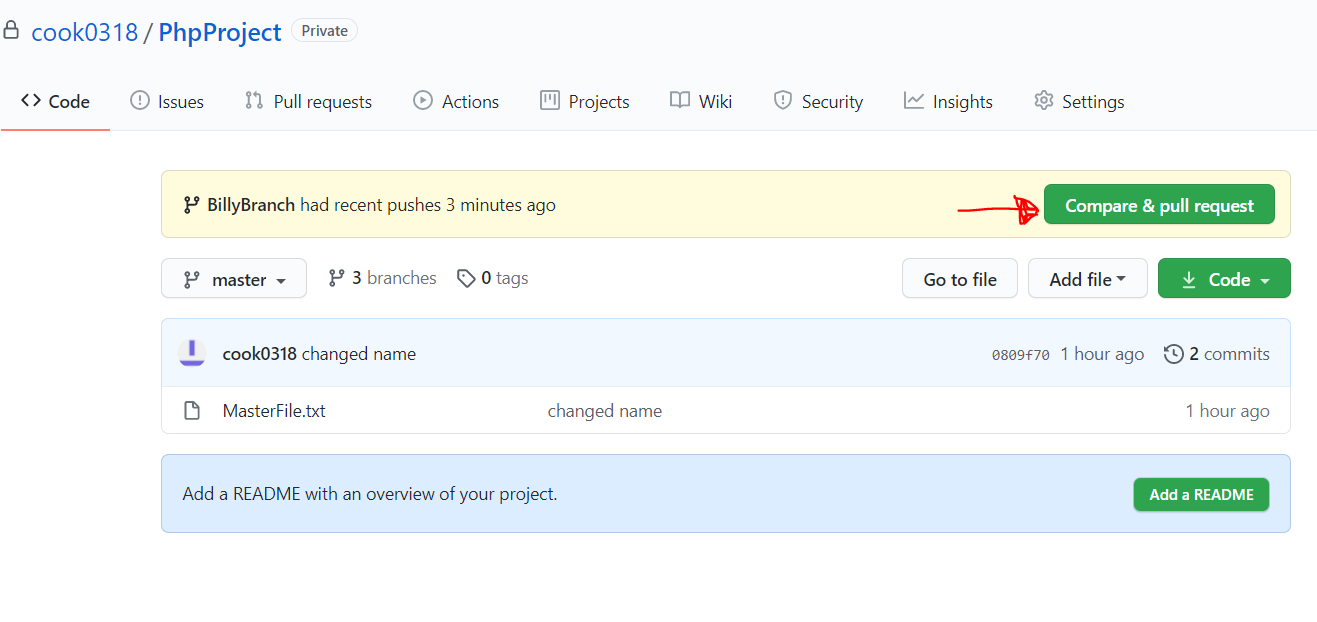
You will have to make a pull request.

This is when you *request* that your code gets *pulled* into the origin/master (you are also technically “pushing” to master...I don’t understand the wording lol)

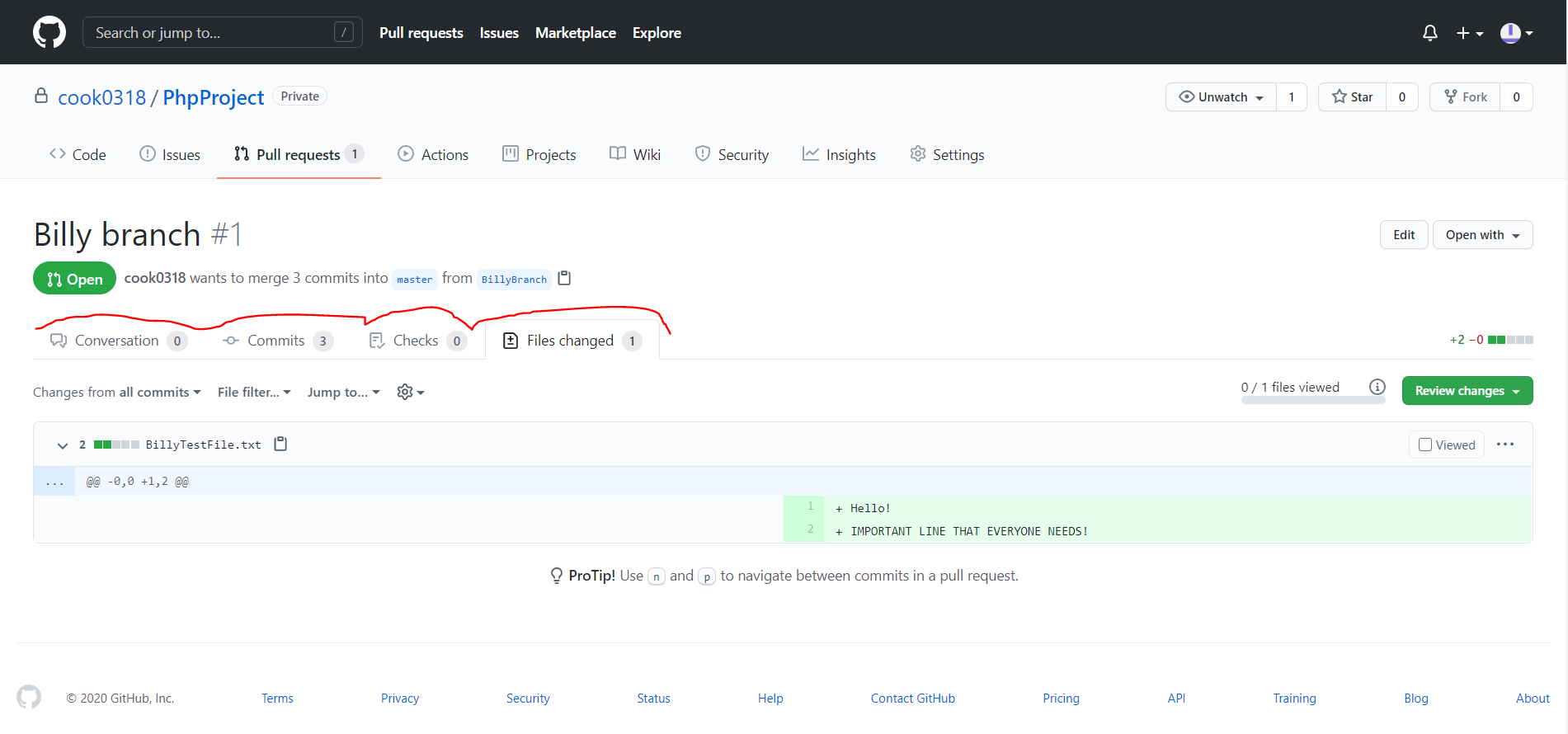
So let’s say you are on BillyBranch and you have BillyTestFile.txt which someone needs.



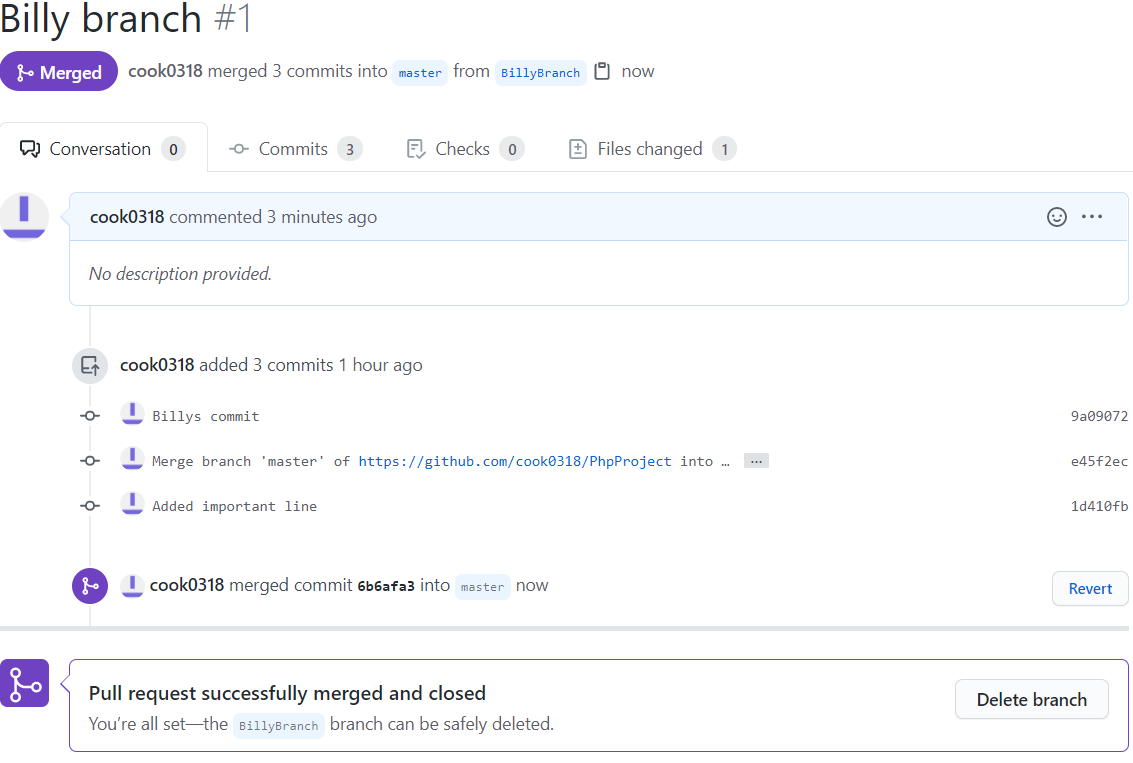
Go to Github and click Compare and Pull Request. Then, Click Create Pull Request.



Now, others can view your request and approve it, or comment on it if they notice something wrong or something that needs changing. It’s up to you guys whether we need to get approval, we can have a rule where either 1 or both the other people have to approve it.



Once it get merged, we are given this screen:



You can delete the branch at this point. This is because **origin/master** is now equal to whatever your code was. So now, if it hasn’t automatically already, run ‘git checkout master’, and update it to be in line with origin/master with ‘**git pull origin master**’. Then you can make a new branch and continue adding your changes.

**NOTE:** I’m not sure how this works if you want to just add 1 file. I think you have to pull your whole folder in.

### **Someone else has done the step above, and I need to update MY branch to include their new code.**

So **origin/master** is ‘newer’ than your local master. And any new changes on **origin/master** are not on your local BillyBranch.

git checkout master // checkout your local master

git pull origin master // **pull** all changes from origin/master into local master

Masters are now up to date.

git checkout BillyBranch // checkout your code branch

git merge master // merge your **local** **master** into your code branch

## **MERGE CONFLICT: I’m trying to make a pull request and there is a merge conflict/I’m trying to do ‘git merge master’ as above and there is a merge conflict.**

Ok this is where I’m least confident in so it may take some hacky solutions like copy/paste and sending files on discord.  
  
But a merge conflict is basically when you are trying to merge two branches together and git doesn’t know which one is “right”.

If shit hits the fan while merging just exit out and write **‘git merge --abort’**.

Example:

You make a local master and make a branch. You add a line (line 2), but you change Line 1 as well. Git looks at this and doesn’t know which to take.

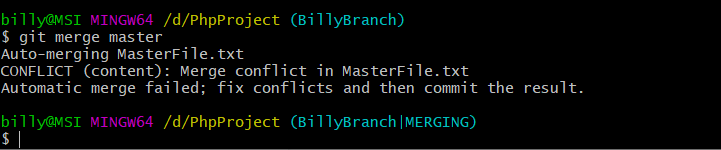
Origin/Master MyCodeBranch

TestFile1.txt Testfile.txt

Line 1: Hello. Line 1: Hello Wei Gong.

+Line 2: Goodbye.

(In this example, the branches are reversed--I added a line to master instead)



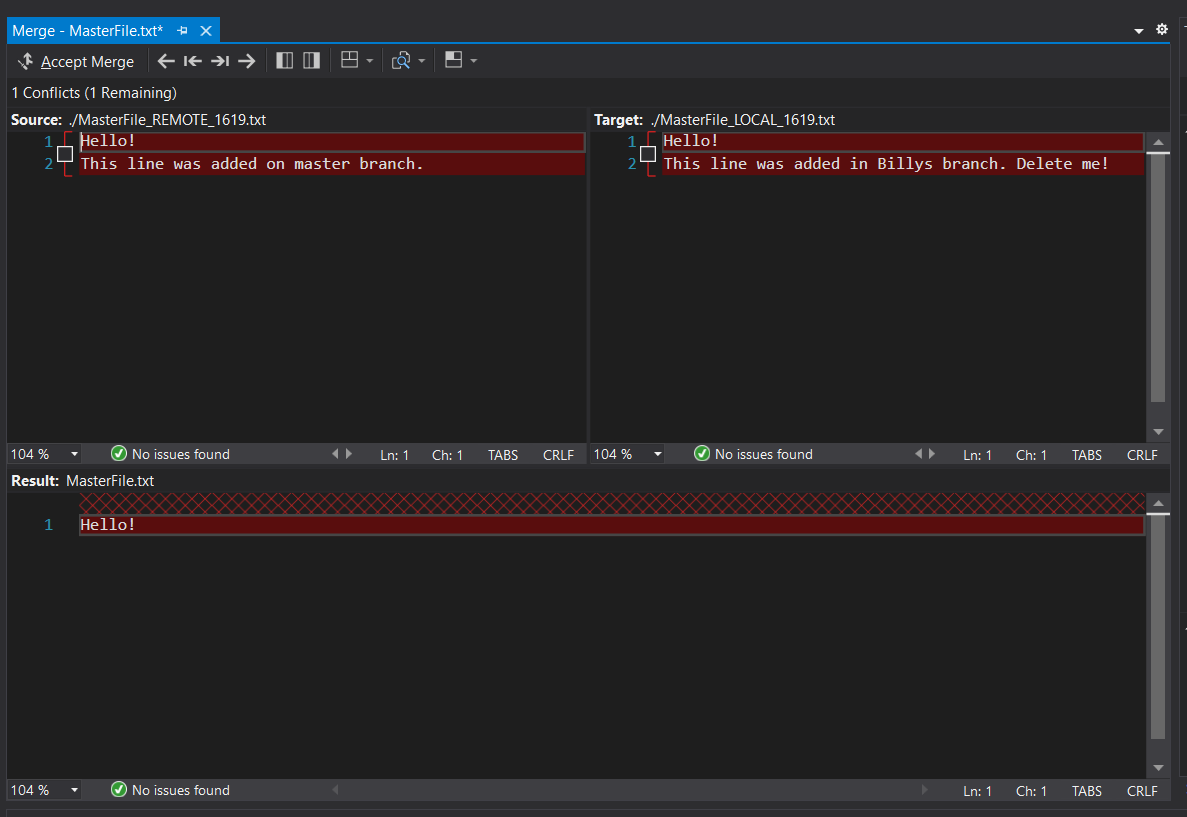
First, ensure Visual studio is your merge tool, see prerequisites.

Type ‘git mergetool’

You will probably get a message saying

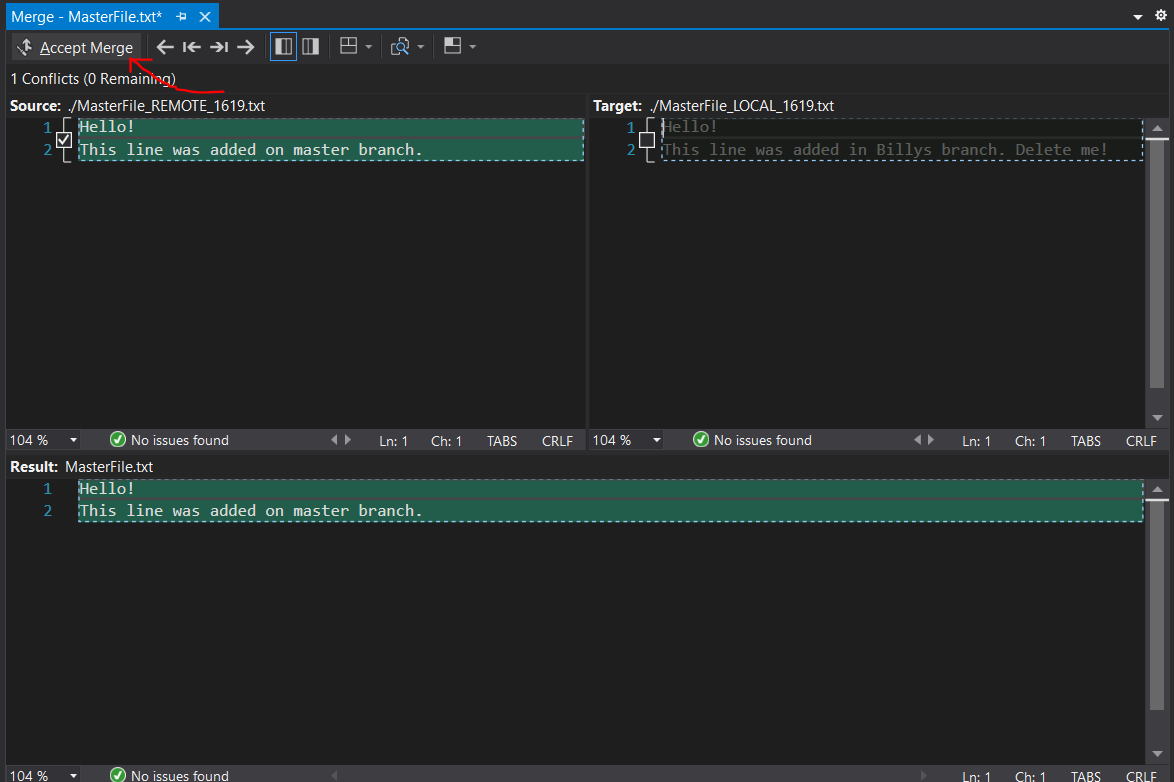
**Blah  
...Blah  
…Blah  
Hit return to start merge resolution tool (vsdiffmerge):**

Click enter and Visual Studio should open.

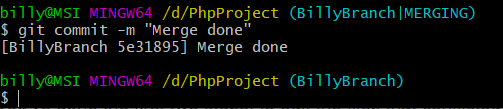


If you are on BillyBranch, and want to merge your local master in, your **Source (top left)** is master and **Target (top right)** is BillyBranch.

Click on the checkbox you want to keep (or click both) and either continue looking for the next conflicts, or click **Accept Merge**



After this, all files will be changed, you just have to commit the changes.



Merge conflict resolved.