

Granular Flow Simulation Manual

Connor Murphy

Dr. Jifu Tan, Northern Illinois University, jifutan@niu.edu

Dr. Nicholas Pohlman, Northern Illinois University, npohlman@niu.edu

Contents

1	Software Components	2
1.1	ParaView	2
1.2	LIGGGHTS Solid Solver	2
1.3	NIU_Undergrad_Research GitHub Repository	2
1.4	Setup	2
1.4.1	Prerequisite Setup	3
1.4.2	Basic Dependencies	3
2	Walk Through	3
2.1	Running a simulation	3
2.2	Data Acquisition	4
3	Maintenance Example	4
3.1	Keeping your repository up to date	4
3.2	Making a Development Branch	6
3.2.1	Creating a development branch via terminal interface	7
3.2.2	Creating a development branch via GitHub.com web interface	9
3.3	Making a commit	10
3.4	Pushing your changes	10
3.5	Making a pull request	12
3.5.1	Making a pull request to merge one branch into another	12
3.5.2	Making a pull request to merge your fork into the main repository	15

1 Software Components

This project has five main software/tools the user will use. These are listed below.

1.1 ParaView



ParaView is an open-source data visualization and analysis software widely used for simulation visualization. It is multi-platform, allowing for users to utilize it on almost any kind of operating system. The link to ParaView's website is given here: [ParaView Website](#)

1.2 LIGGGHTS Solid Solver

LIGGGHTS (LAMMPS Improved for General Granular and Granular Heat Transfer Simulations) is a molecular dynamics solver library that is used by this project to simulate granular flow of pellet-shaped particles. It is massively parallel, which allow this simulation to be run on a large number of processors. The installation steps for LIGGGHTS are described in depth in its documentation, the link to which can be found here: [LIGGGHTS Installation Steps](#)

1.3 NIU_Undergrad_Research GitHub Repository

All the LIGGGHTS simulation scripts, MATLAB sorting/plotting scripts, and pellet data files are stored in this GitHub repository. [Project Repository](#)

1.4 Setup

This section will discuss setting up the project to run on your local machine. It assumes you already have a computer with windows 10 or 11 operating system. If you are already familiar with Windows Subsystem for Linux (WSL) and Ubuntu, feel free to skip the the Prerequisite Setup section. If you are already using Linux as your host operating system you should also be able to skip ahead to installing required dependencies though your workflow may be different. If you are working on a pre-M1 Mac your experience should be similar to that of Linux. M1 Macs have not been tested and because of the different processor architecture is unlikely to work.

1.4.1 Prerequisite Setup

Before we can setup the project specific dependencies, Ubuntu must be installed. This project used Ubuntu on WSL (Windows Subsystem for Linux). Documentation on how to install Ubuntu can be found on: [Installing Ubuntu on WSL2](#). We recommend using Ubuntu version 18.04 LTS or 20.04 LTS. Make sure you also complete the steps on Ubuntu's web-page for installing Windows Terminal. from the Microsoft Store

1.4.2 Basic Dependencies

You will need to install the basic dependencies that are used by the project. Please run these two commands in your terminal:

```
sudo apt-get update

sudo apt-get install -y \
git \
build-essential \
autoconf \
libtool \
libmpich-dev \
libssl-dev \
wget \
pkg-config \
python3-dev \
python3-numpy \
libosmesa6-dev \
libgl1-mesa-dev \
libtbb-dev
```

2 Walk Through

2.1 Running a simulation

The repository where all the code is held needs to be cloned to your computer. In the Ubuntu terminal, use the following command:

```
git clone https://github.com/Mectr0/NIU_Undergrad_Research.git
```

Once LIGGGHTS is built, an executable named `lmp_auto` should exist in `/LIGGGHTS-PUBLIC/src`. Move this over to `NIU_Undergrad_Research/scripts`. This directory exists in the repository that was just cloned. With this executable, most of the simulation scripts in this directory can now be run. The main simulation script is `in.fullbin` and can be run using:

```
mpirun -np X lmp_auto < in.multisphere
```

Where X is the number of ranks to be run on. Documentation for all of the commands used in `in.fullbin` can be found on LIGGGHTS Documentation, but most commands have a brief description of what they do in the script itself.

2.2 Data Acquisition

A command exists in `in.fullbin` that creates a custom dump file with id values, velocity, position, and orientation for each pellet. The current name given for this file is `dump.pellet_X.0D` where X is the value of the exit height in pellet diameters (adjustable in the simulation script). To use this dump file, move it over to the `NIU_Undergrad_Research/PostProcessing/Sorting_Pellets` directory after the simulation has finished. The dump file can now be used by the Matlab file `SortingScript.m` in this directory. Documentation on how to use this sorting script is found within the script itself when opened in Matlab. Once a `.mat` file is created by `SortingScript.m` and moved to `NIU_Undergrad_Research/PostProcessing/Data_Plotting`, it can be used by `PlottingData.m`. This script generates of the data plots used so far.

3 Maintenance Example

Most maintenance concerning the project is how to properly maintain your GitHub repository and code. The following sections will cover some general practices used to keep your GitHub repository up to date. If you haven't yet finished the prerequisite setup you will need to see that section to get your GitHub account set up and create your own fork of the BloodFlow repository.

3.1 Keeping your repository up to date

It is important to keep your repository up to date with the main BloodFlow repository. The easiest way to do this is through the use of the Fetch and Merge button on the GitHub web interface. It's as simple as going to your repository and click on the Fetch and Merge button.

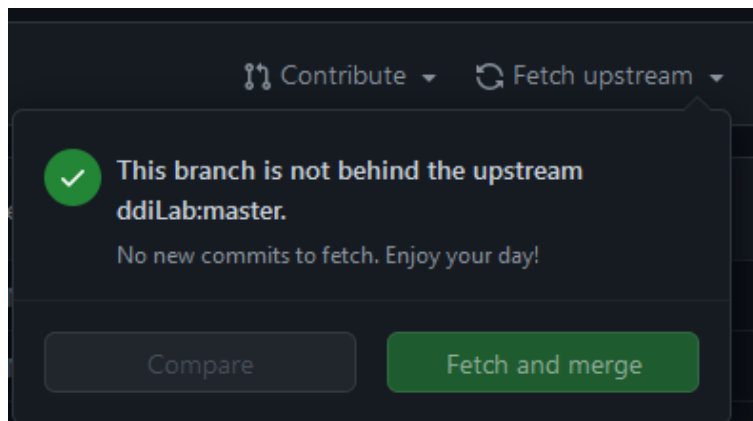


Figure 1: Example of Fetching upstream when you are already up-to date.

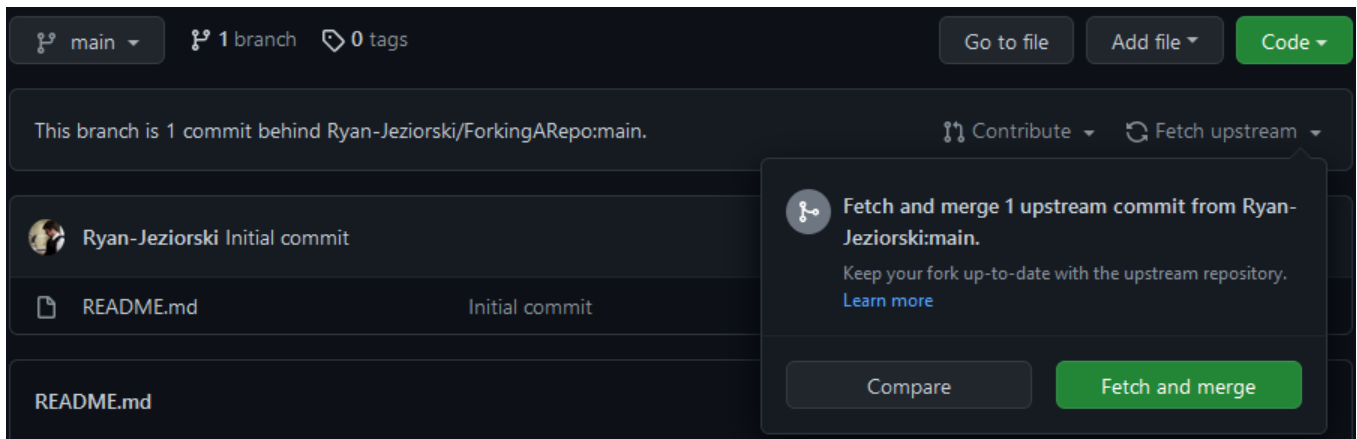


Figure 2: Example of Fetching upstream when you are not up-to date and there are no conflicts.

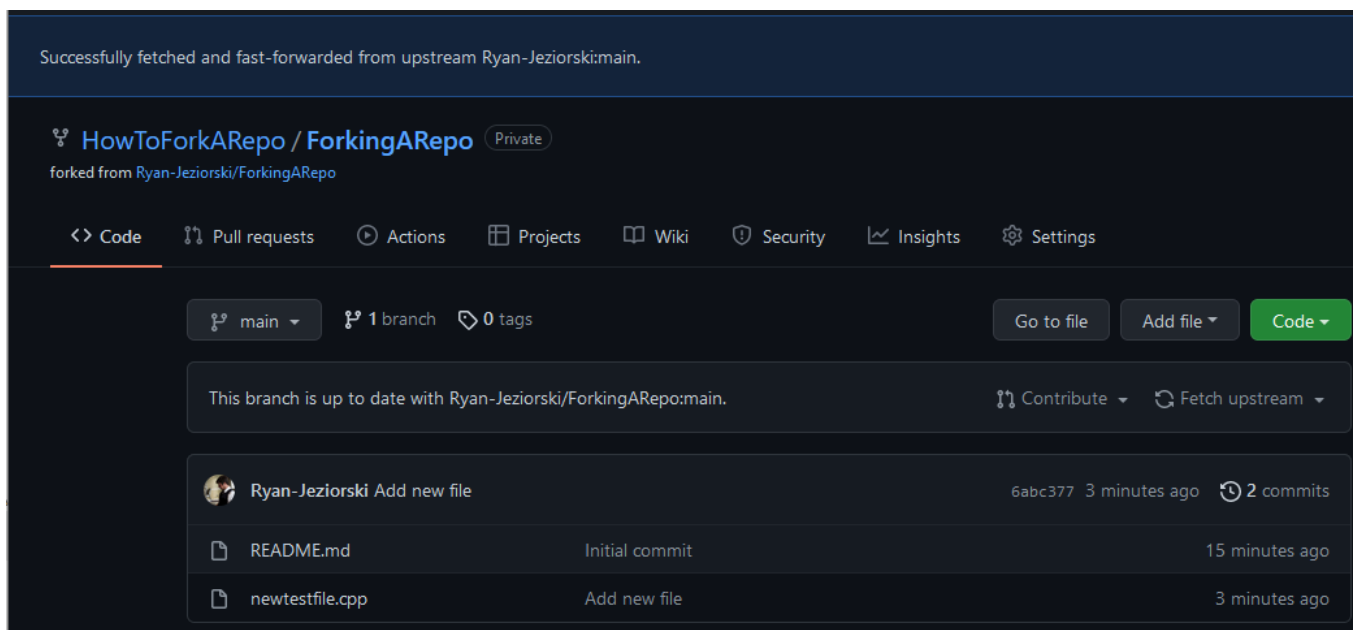


Figure 3: After you've clicked on "Fetch and Merge" you will be notified that it has been successful.

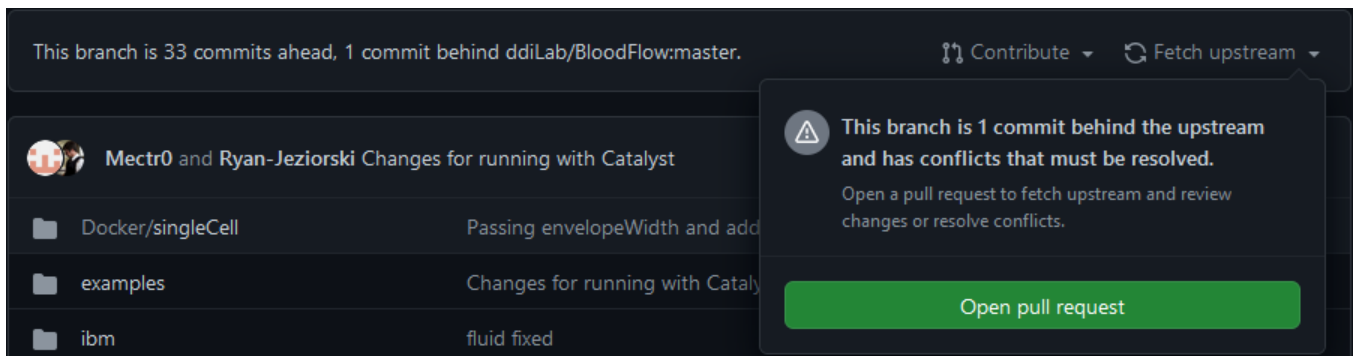


Figure 4: Example of Fetching upstream when both you and the main branch have made changes and there are conflicts. Notice how the branch is 33 commits ahead of and 1 commit behind ddiLab/BloodFlow:master. Solving this is a quite complicated issue and it is recommended that you reach out to your faculty mentors in this instance as it's quite easy to get tangled up here and is one of the only instances where the loss of work is possible. So be careful.

3.2 Making a Development Branch

Another best practice for maintaining your repository on GitHub is making use of branches. This allows you to keep the development and stable versions of your code separate. This section will go over how to create a development branch as well as when you should merge your development branch into your main branch. There are two ways to create a new branch with Git, we will cover both here though do note that the preferred way is generally via the terminal. Before we begin creating a new branch I want to introduce three commands that are useful, they are:

```
#This shows the status of the Git repository
git status
```

```
#This shows all of the branches that are local to your machine
git branch
```

```
#This is the command you can use to switch branches
git checkout [name_of_branch_to_switch_to]
```

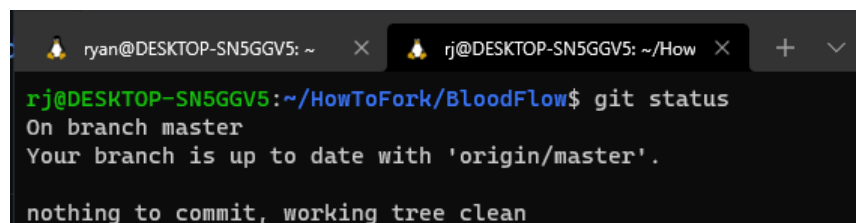
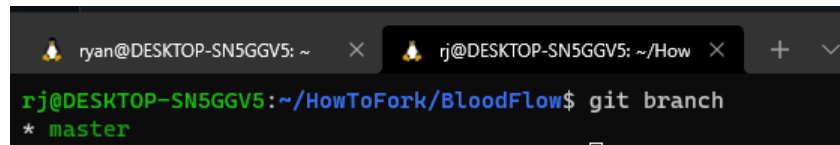
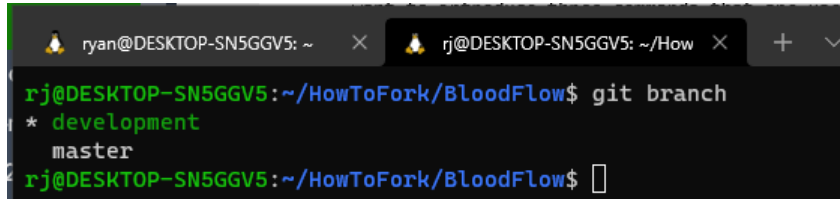


Figure 5: Example of git status command



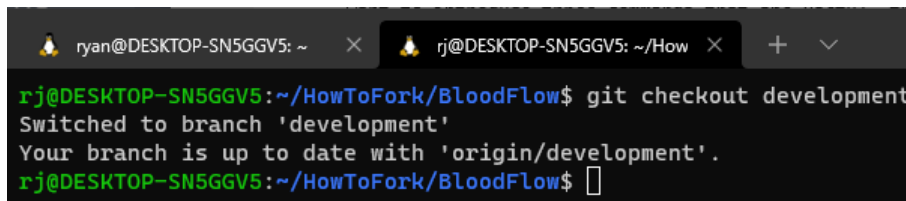
```
ryan@DESKTOP-SN5GGV5: ~  
rj@DESKTOP-SN5GGV5: ~/HowToFork/BloodFlow$ git branch  
* master
```

Figure 6: Example of git branch command with only one branch



```
ryan@DESKTOP-SN5GGV5: ~  
rj@DESKTOP-SN5GGV5: ~/HowToFork/BloodFlow$ git branch  
* development  
master  
rj@DESKTOP-SN5GGV5: ~/HowToFork/BloodFlow$
```

Figure 7: Example of git branch command with multiple branches



```
ryan@DESKTOP-SN5GGV5: ~  
rj@DESKTOP-SN5GGV5: ~/HowToFork/BloodFlow$ git checkout development  
Switched to branch 'development'  
Your branch is up to date with 'origin/development'.  
rj@DESKTOP-SN5GGV5: ~/HowToFork/BloodFlow$
```

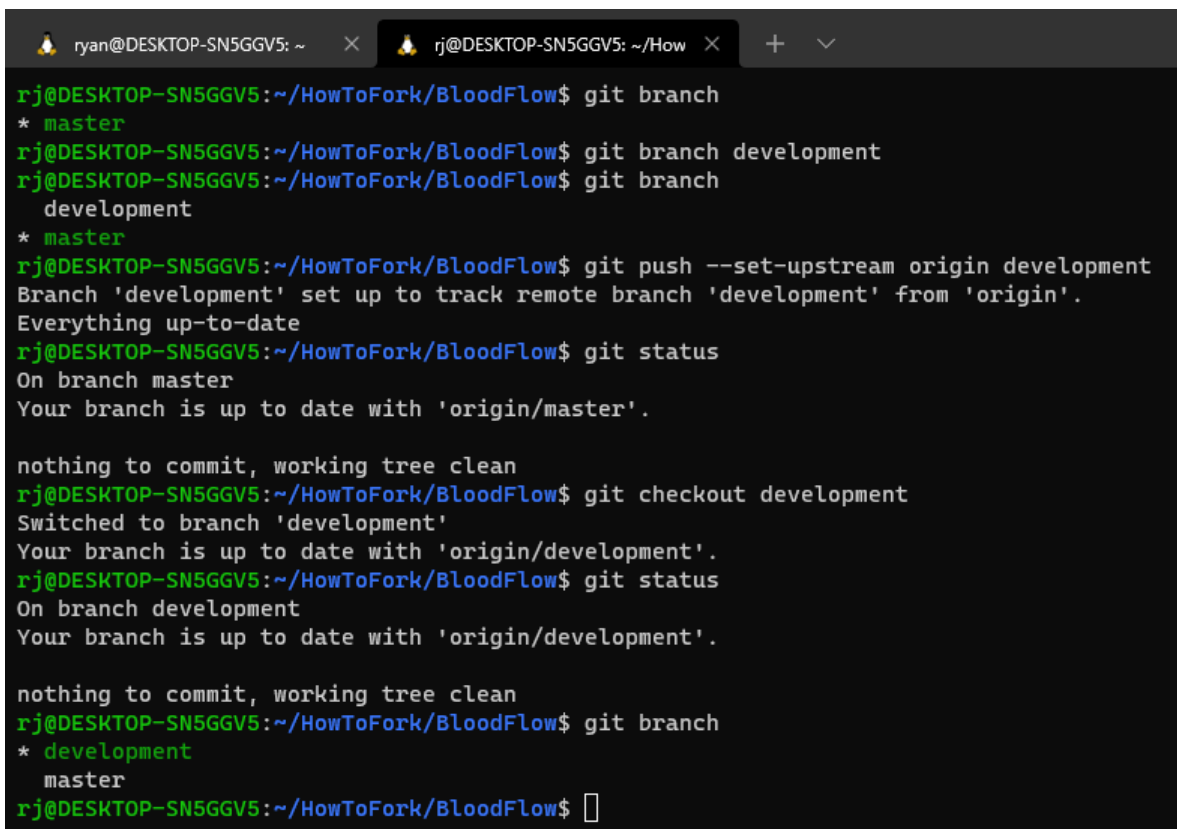
Figure 8: Example of git checkout command

3.2.1 Creating a development branch via terminal interface

The easiest and fastest way to create a new branch for your repository is via the terminal interface using the commands shown below.

1. Method 1

```
# This creates a new branch named development  
git branch development  
  
# This shows all of the branches  
git branch  
  
# This switches to the newly created branch  
git checkout development  
  
# This pushes the new branch to the web repository  
# The push command will be covered in more detail in that section  
git push --set-upstream origin development
```

```
ryan@DESKTOP-SN5GGV5: ~  
rj@DESKTOP-SN5GGV5: ~/How  
rj@DESKTOP-SN5GGV5:~/HowToFork/BloodFlow$ git branch  
* master  
rj@DESKTOP-SN5GGV5:~/HowToFork/BloodFlow$ git branch development  
rj@DESKTOP-SN5GGV5:~/HowToFork/BloodFlow$ git branch  
development  
* master  
rj@DESKTOP-SN5GGV5:~/HowToFork/BloodFlow$ git push --set-upstream origin development  
Branch 'development' set up to track remote branch 'development' from 'origin'.  
Everything up-to-date  
rj@DESKTOP-SN5GGV5:~/HowToFork/BloodFlow$ git status  
On branch master  
Your branch is up to date with 'origin/master'.  
  
nothing to commit, working tree clean  
rj@DESKTOP-SN5GGV5:~/HowToFork/BloodFlow$ git checkout development  
Switched to branch 'development'  
Your branch is up to date with 'origin/development'.  
rj@DESKTOP-SN5GGV5:~/HowToFork/BloodFlow$ git status  
On branch development  
Your branch is up to date with 'origin/development'.  
  
nothing to commit, working tree clean  
rj@DESKTOP-SN5GGV5:~/HowToFork/BloodFlow$ git branch  
* development  
master  
rj@DESKTOP-SN5GGV5:~/HowToFork/BloodFlow$
```

Figure 9: Example showing the commands in action

2. Method 2 [recommended]

```
# By adding the -b optional argument to the checkout command we  
# can both create and switch to the new branch with one command  
git checkout -b development
```

```
# This pushes the new branch to the web repository  
git push --set-upstream origin development
```

```
ryan@DESKTOP-SN5GGV5: ~  
rj@DESKTOP-SN5GGV5: ~/HowToFork/BloodFlow$ git checkout -b development  
Switched to a new branch 'development'  
rj@DESKTOP-SN5GGV5:~/HowToFork/BloodFlow$ git push  
fatal: The current branch development has no upstream branch.  
To push the current branch and set the remote as upstream, use  
  
git push --set-upstream origin development  
  
rj@DESKTOP-SN5GGV5:~/HowToFork/BloodFlow$ git push --set-upstream origin development  
Total 0 (delta 0), reused 0 (delta 0)  
remote:  
remote: Create a pull request for 'development' on GitHub by visiting:  
remote:      https://github.com/Ryan-Jeziorski/BloodFlow/pull/new/development  
remote:  
To github.com:Ryan-Jeziorski/BloodFlow.git  
* [new branch]      development -> development  
Branch 'development' set up to track remote branch 'development' from 'origin'.  
rj@DESKTOP-SN5GGV5:~/HowToFork/BloodFlow$
```

Figure 10: Example showing the commands in action

3.2.2 Creating a development branch via GitHub.com web interface

The other way to create a new branch is via the GitHub web interface, while this has the advantage of being able to be done through a web browser and with a GUI it is a slightly longer process and it is usually preferable to make a new branch through the terminal. It is being shown here for completeness.

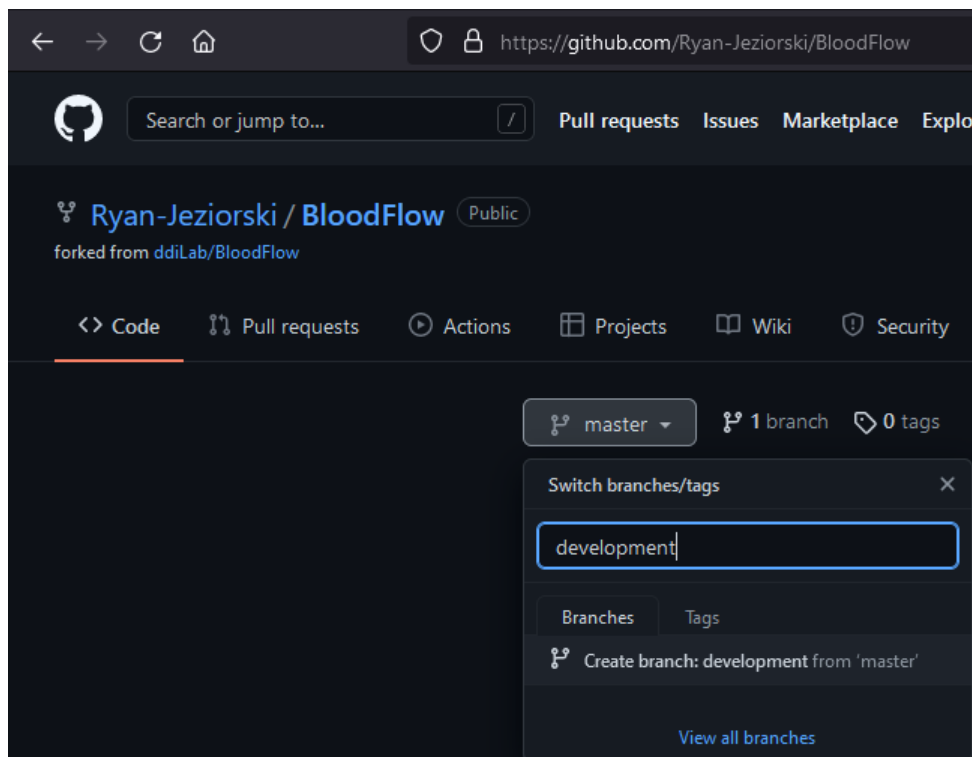


Figure 11: Example showing the commands in action

3.3 Making a commit

Making a commit to your repository can be thought of as similar to saving a file. Before a commit has been made none of the changes that you've made to any of the files in the repository should not be seen as permanent. A commit is like saying the changes that I have made to this file work and I wish to save them. This is a very useful feature as before you've committed you can undo any of the changes you've quite easily allowing you to always revert back to a working version of the project. This is very useful for testing out new changes without fear of ruining the project.

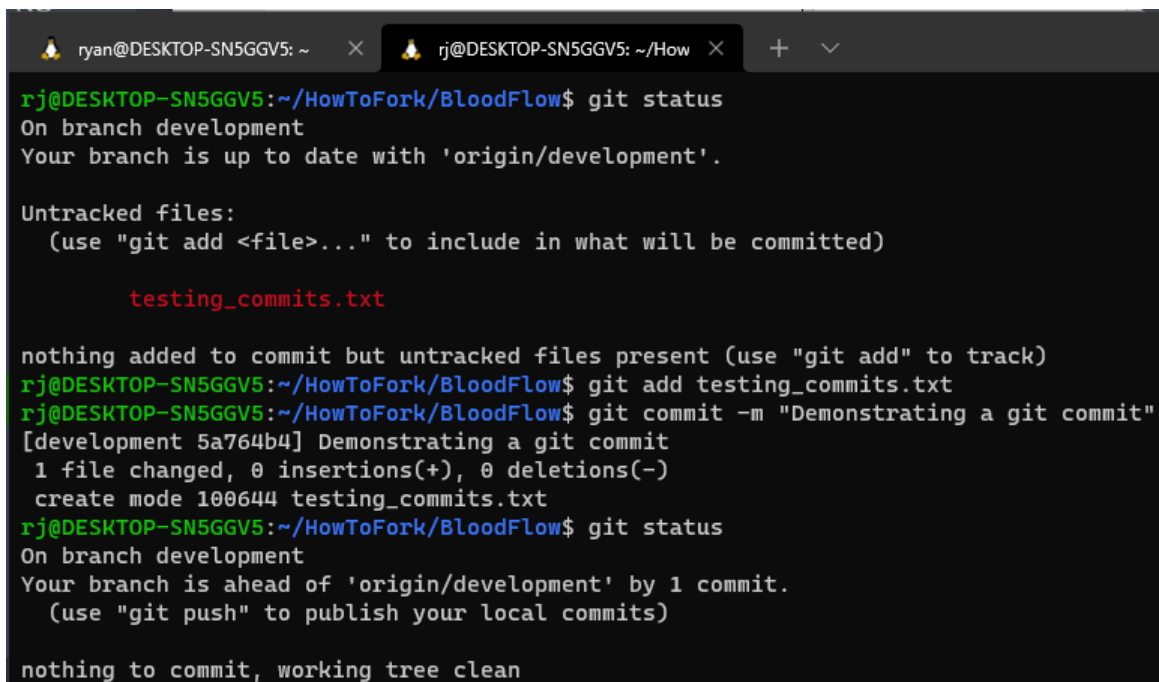
```
# This is the format of the git add and git commit commands.
# Note that with the git add command you can add multiple file names.

git add [file_name(s)]

# The quotation marks are important and part of the syntax for the command.
# Also note the '-m' this stands for message and is also needed.

git commit -m "Your commit message here"
```

You will notice below that when you use the git status command you will be notified of files that have been changed or added in red. You can then add the files with the git add command. And finally using the git commit command you can commit the changes to the repository. Afterwards you can run the git status command again to see that you have no uncommitted changes.



```
ryan@DESKTOP-SN5GGV5: ~
rj@DESKTOP-SN5GGV5: ~/HowToFork/BloodFlow$ git status
On branch development
Your branch is up to date with 'origin/development'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    testing_commits.txt

nothing added to commit but untracked files present (use "git add" to track)
rj@DESKTOP-SN5GGV5:~/HowToFork/BloodFlow$ git add testing_commits.txt
rj@DESKTOP-SN5GGV5:~/HowToFork/BloodFlow$ git commit -m "Demonstrating a git commit"
[development 5a764b4] Demonstrating a git commit
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 testing_commits.txt
rj@DESKTOP-SN5GGV5:~/HowToFork/BloodFlow$ git status
On branch development
Your branch is ahead of 'origin/development' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
```

Figure 12: Example of how to make a commit

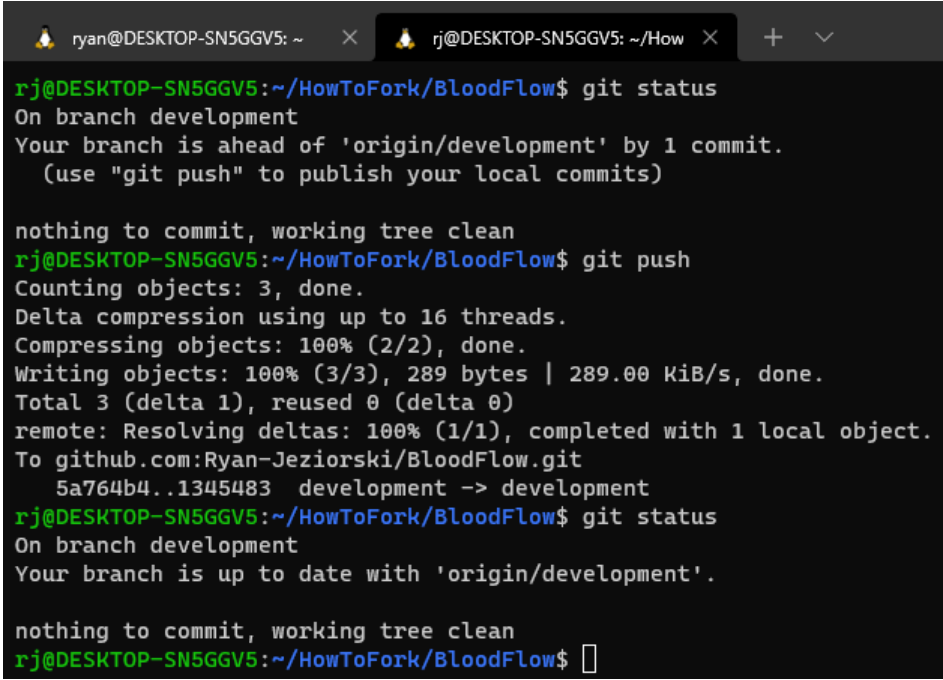
3.4 Pushing your changes

So now you've successfully made a change, and committed those changes to your local repository. There is one more step in making those changes permanent, currently the changes you have

made exist only on your local machine. Should your computer become incapacitated those changes would be lost. Fear not though, by pushing the changes to the GitHub servers those changes will become more permanent, as well as shareable. It's quite easy to make a push the figure below will demonstrate.

```
# This is the format of the git push command.
# You may notice the command from the workflow for creating a new branch

# Simply make sure you are on the branch you wish to push
git push
```

A terminal window with two tabs. The first tab is titled 'ryan@DESKTOP-SN5GGV5: ~' and the second is 'rj@DESKTOP-SN5GGV5: ~/How'. The terminal shows the following commands and output:

```
rj@DESKTOP-SN5GGV5:~/HowToFork/BloodFlow$ git status
On branch development
Your branch is ahead of 'origin/development' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
rj@DESKTOP-SN5GGV5:~/HowToFork/BloodFlow$ git push
Counting objects: 3, done.
Delta compression using up to 16 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 289 bytes | 289.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To github.com:Ryan-Jeziorski/BloodFlow.git
   5a764b4..1345483  development -> development
rj@DESKTOP-SN5GGV5:~/HowToFork/BloodFlow$ git status
On branch development
Your branch is up to date with 'origin/development'.

nothing to commit, working tree clean
rj@DESKTOP-SN5GGV5:~/HowToFork/BloodFlow$
```

Figure 13: Example of how to push

You should now be able to see the changes that you've made on the GitHub web page by following the sequence of images below.

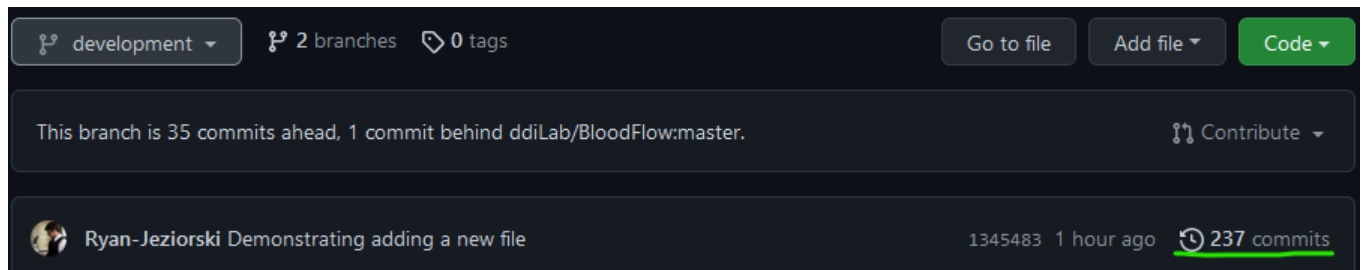


Figure 14: Click on the commit button

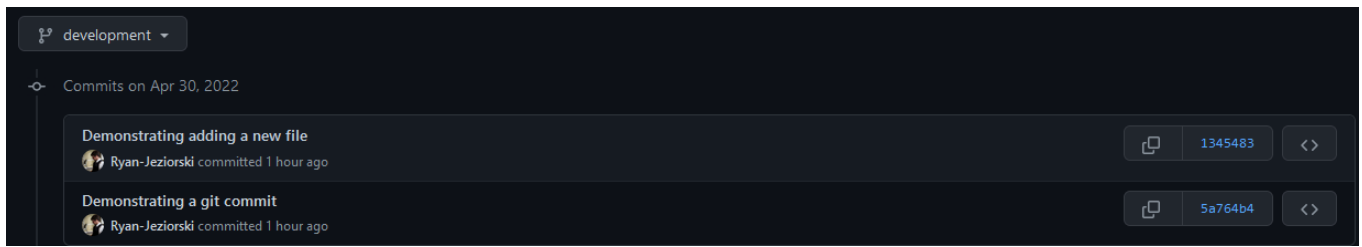


Figure 15: You can now see the commit history

3.5 Making a pull request

Once you have made all the changes necessary to add that shiny new feature to the project we can merge the development branch into the main branch. You will do this with a pull request. In this manual we will talk about two types of pull requests. The first is making a pull request to merge your development branch into your main branch. The second will be making a pull request to merge your changes on your fork into the main BloodFlow repository.

3.5.1 Making a pull request to merge one branch into another

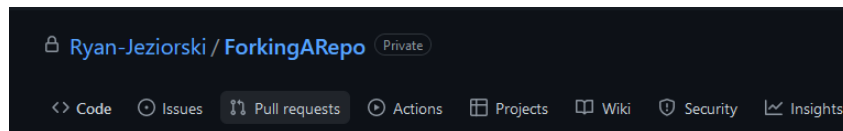


Figure 16: First on your repository's main page, find and click the pull request icon

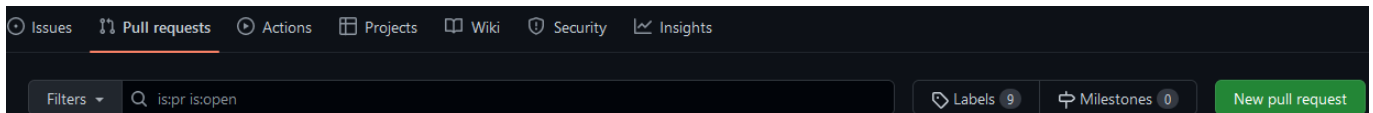


Figure 17: You will arrive on the pull request page, you'll need to click "new pull request"

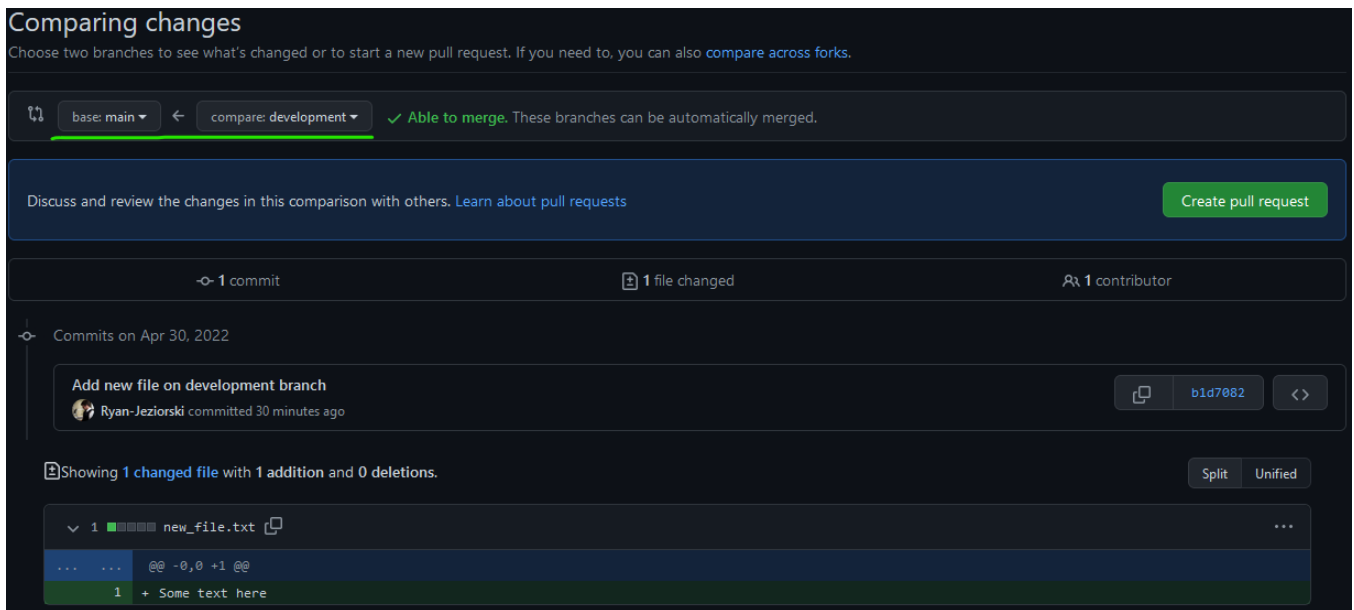


Figure 18: Notice the direction of the arrow, the final destination of the pull request is on the left, and the branch containing the changes you wish to merge are on the right. Once you've selected the branches you wish click on the "Create pull request" button." Note, you can also see the changes that are going to be made on the bottom of the screen.

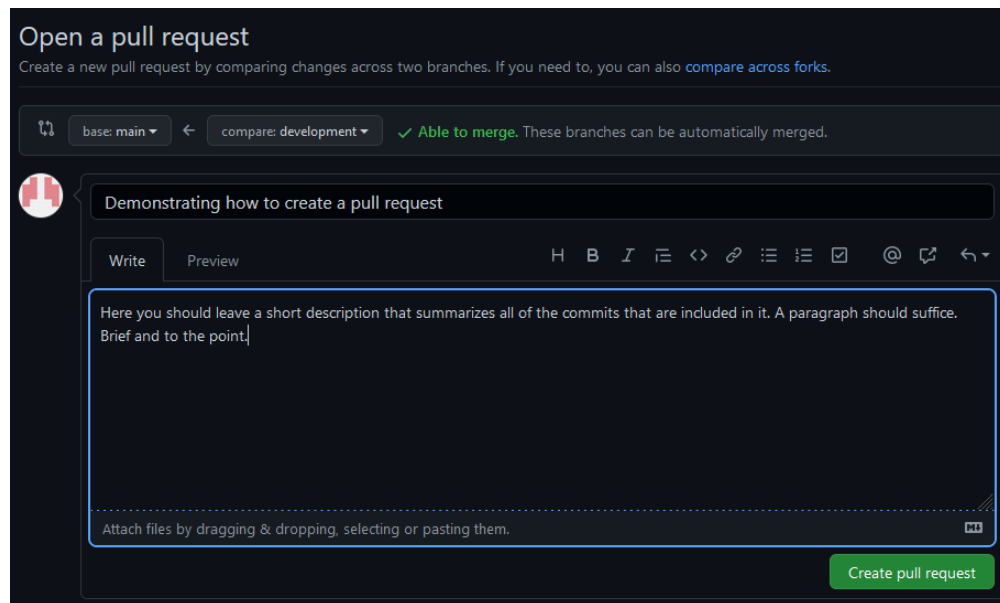


Figure 19: Follow the instructions in the image and then click create pull request.

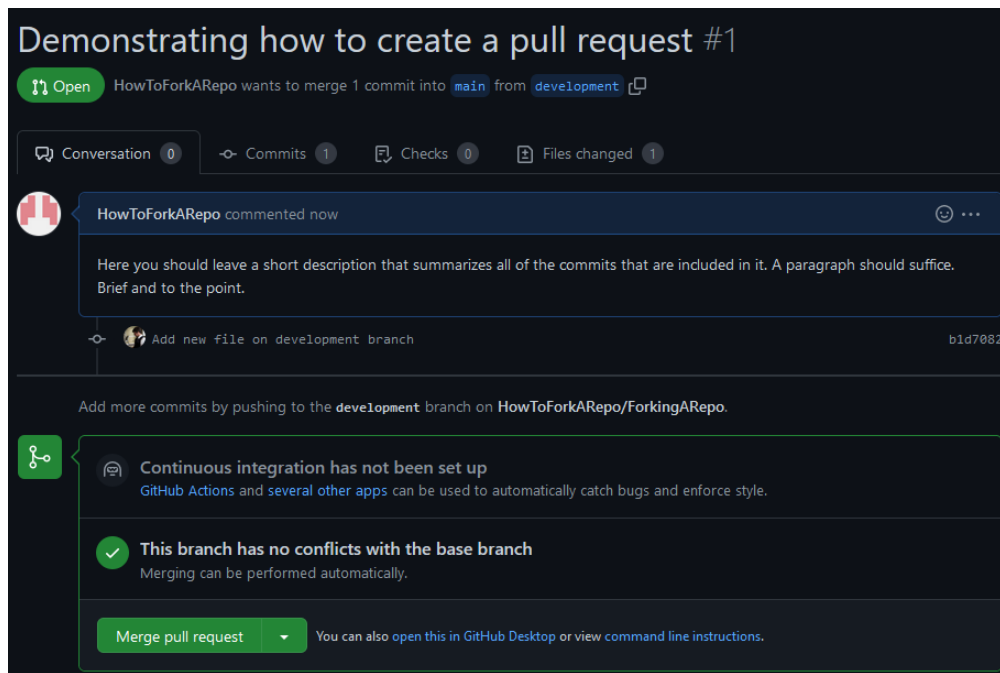


Figure 20: Now you'll need to click on the merge pull request button.

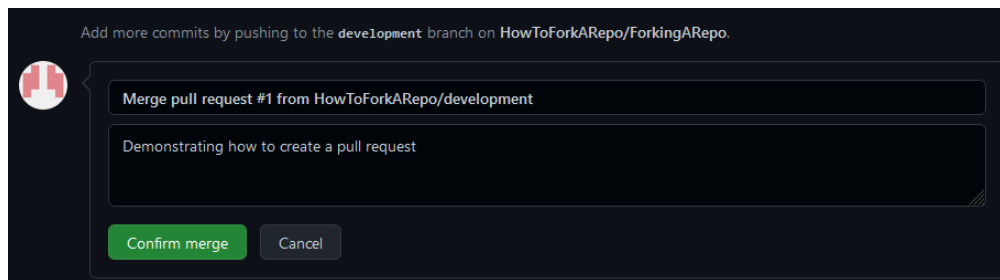


Figure 21: Here you can change the message should you wish, it defaults to your previous message, click on Confirm merge.

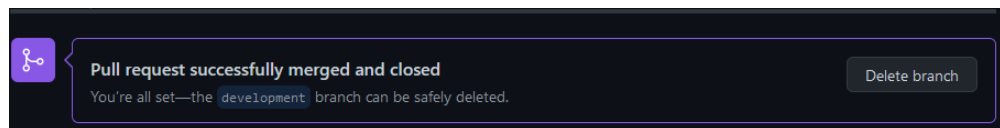


Figure 22: You should now delete the old branch when prompted.

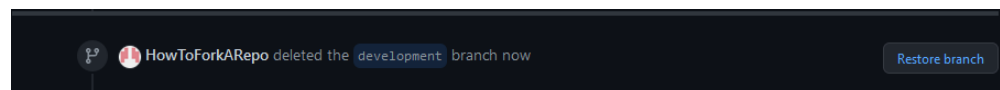


Figure 23: If you have made a mistake and deleted the branch too soon you can still restore it, otherwise you will now need to follow the steps from above to create a new branch and continue working.

3.5.2 Making a pull request to merge your fork into the main repository

Making a pull request into the main repository is done in much the same way as a pull request between branches. The main difference between the steps from above is that now when you are selecting the branches to merge to and from, is that you will need to select the repository for the BloodFlow, and your individual Fork as well

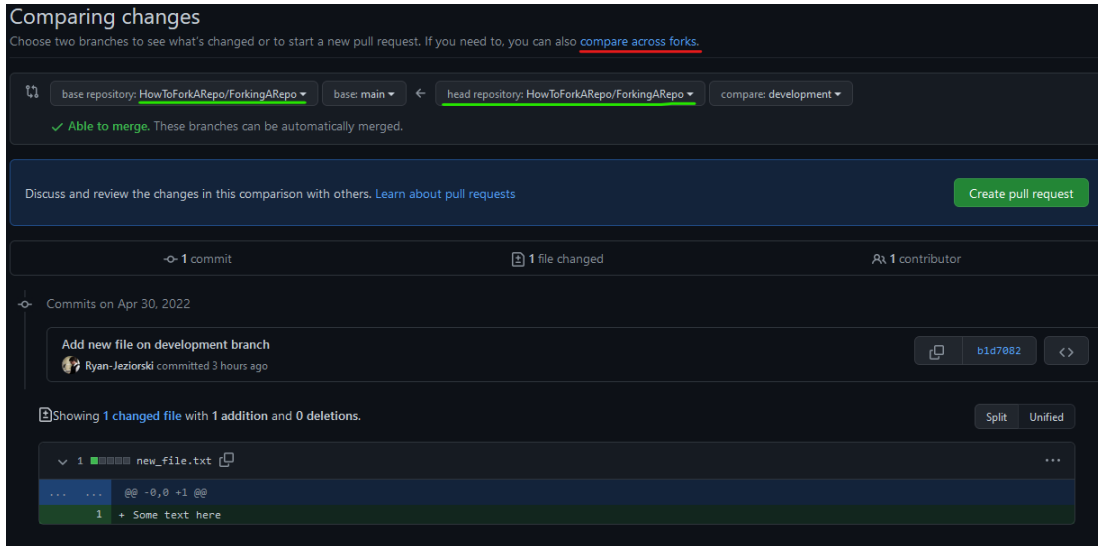


Figure 24: Note: If your screen does not look like this you can click the "compare across forks" button and it will switch for you.

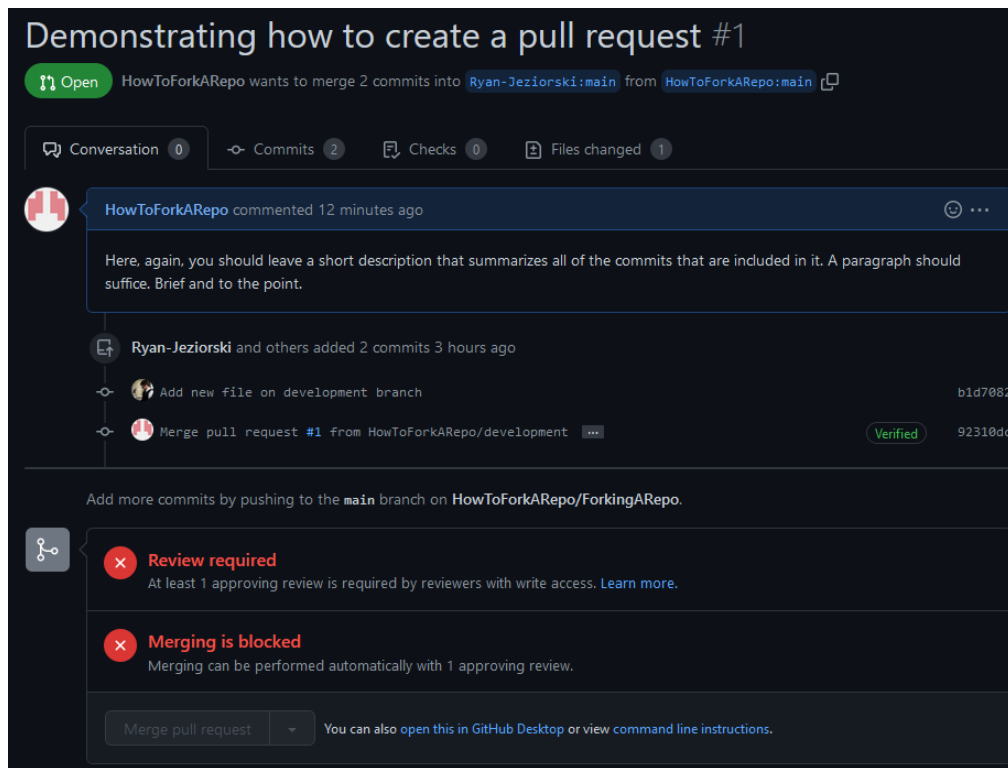


Figure 25: Note once you get to the confirm merge request screen, you will not be able to merge as you do not have permissions to. You will need to notify your team that you have opened a pull request and work with them on getting it merged.