# Hyperiondev

**TASK**

# Version Control II: Pipelines

Visit our website

# Introduction

## TIME TO GET REAL WITH VERSION CONTROL

In past tasks you saw how to manage different versions of the same codebase. In this task we'll look at how git helps ensure multiple developers can work simultaneously on the same project without getting in each other's way. One day you'll be one such developer working on a team project, so it's important to know how it's done. Feel free to refer back to the previous version control tasks for a refresher before getting started.

## CLONING A REPO

You know how to initialise a Git repository from scratch, but in practice, this is very rarely how you start developing. Typically when starting as a software engineer, you'll be tasked with closing one or more issues on a specific repository. We'll get onto exactly what "closing an issue" means, but for the time being let's first focus on getting the repo cloned so that you can start developing.

For this task, you'll be using your last capstone project's repository. If you haven't put it on GitHub yet, go ahead and do that now. Refer to the previous Git task if you need a reminder of how to do it.

For this task we'll be pretending your repo is a team project, and you'll be playing the role of developer, code reviewer, and even project manager. To start off, pretend you're a developer seeing the project for the first time and clone the repo:

```
git clone https://github.com/[username]/[repo].git
```
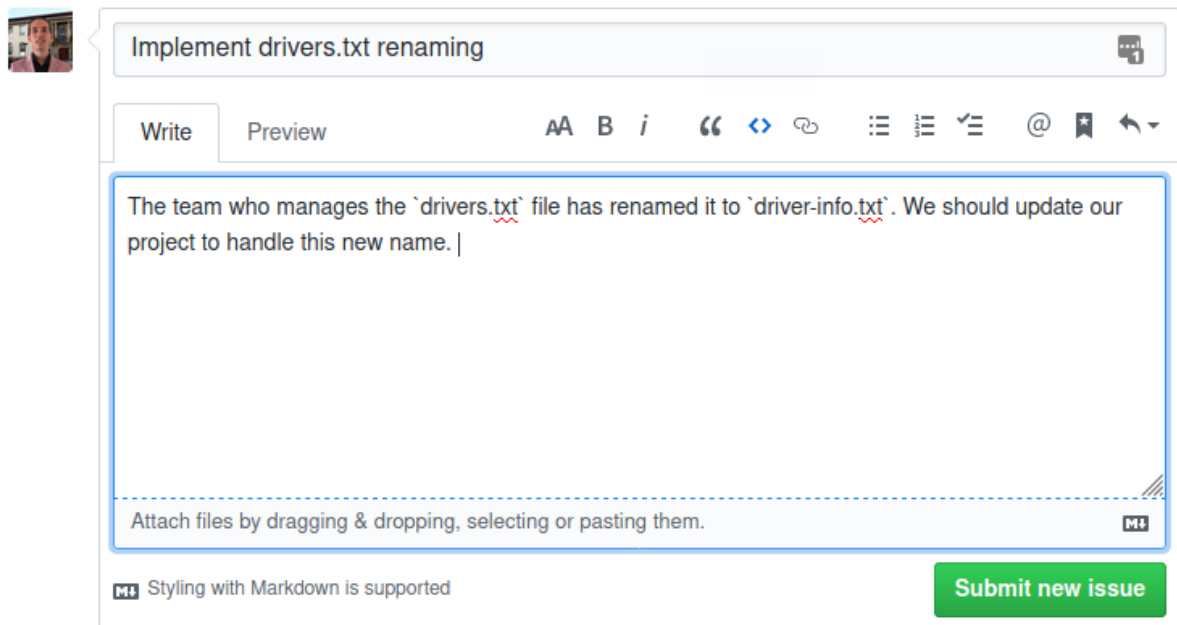
If you were at your job, at this point you'd usually spend a while compiling the code, resolving dependencies and fixing issues on your computer to get the project running. This should now, however, not be a problem since you recently wrote that very code.

## ISSUES

On GitHub (and many other platforms) an issue is a plain-English description of some improvement a codebase needs. This could be a bug fix, a performance improvement, or a new feature. Software engineers almost never do any work unless there's an issue attached to it. This makes sure everyone knows who's doing what and helps line managers understand what's being done to the codebase and what the progress is.

On the GitHub website, go to your repo and click on "Issues" (on the top bar, next to Code). You should see a plain greeting message since your repo doesn't have any issues yet (yay!).

Pretend you're a project manager and you just came back from a meeting where another team asked you to make a change to your project. In order to let your team know what needs to be done, you make an issue on the repo. Click on "New Issue" (top right) and fill in the following details:



When finished, click "Submit" to officially create the issue. You'll see the issue has been given an ID (**#1**).

To the right of your issue description, click on the gear icon next to "Assignees". Type in your username and click on it to assign yourself to the job. If you were a real project manager, you'd be assigning one or more of your team members.

## DOING WORK ON YOUR OWN BRANCH

Now that you (as the developer) have been assigned to the issue, you should implement it. But before doing any coding, be sure to go to the directory and create a new branch. This ensures you don't accidentally work on a co-worker's branch and their code gets overwritten later on. You can name the branch something descriptive, indicating what you'll be doing on it (like `driver-file-fix`), (`hobby-need-more-detail`) or to ensure uniqueness you can name it according to the issue ID (`issue-1`). Each company will have its own policy in this respect. As you may recall, the code below creates the branch `issue-1` and switches to it at the same time.

```
git checkout -b issue-1
```

Now implement the change. As described in the issue, you need to make the program use `driver-info.txt` instead of `drivers.txt`. Be sure to rename the actual file too and run your program to ensure there are no problems with the new change.

Now commit your changes, giving a good description of what you changed, and then push your branch to the remote repo.
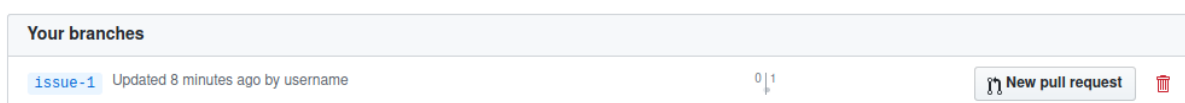
```
git commit -m "Changed program to use driver-info.txt"
git push origin issue-1
```

You can make multiple commits and pushes on your branch as needed until you are satisfied the change has successfully been implemented.
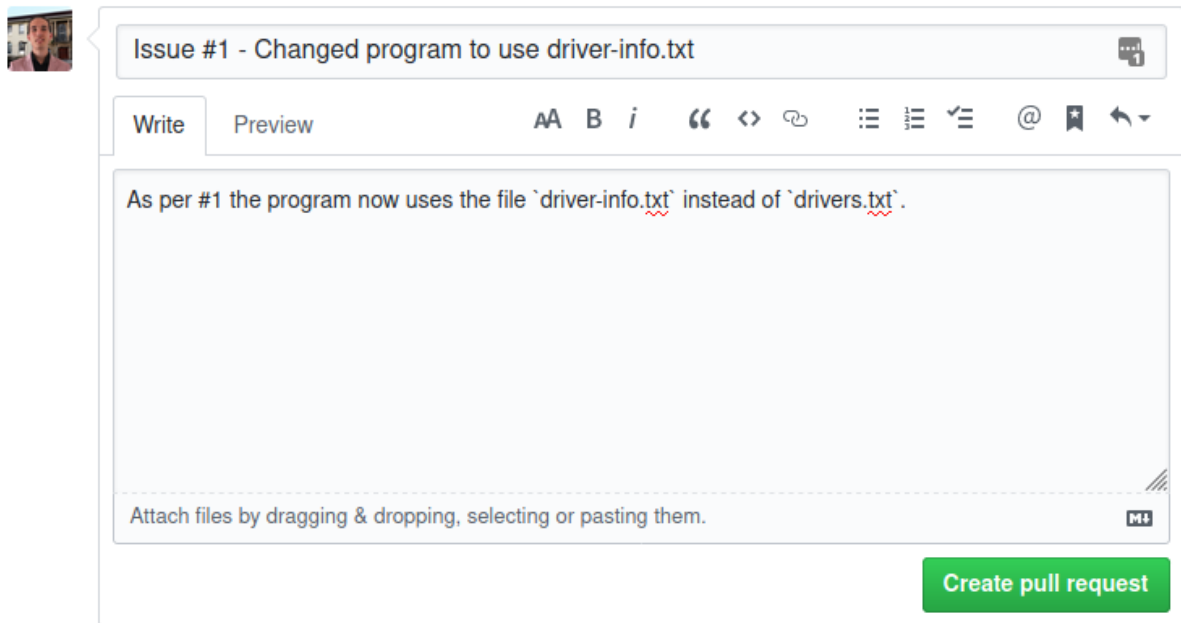

## MAKING A PULL REQUEST

A pull request (PR) is a request asking that your branch be merged with the master branch. If accepted, this will make your code changes part of the main codebase. Depending on how the repo is set up, this might even deploy the new version of the program to your users.

It is the responsibility of the developer who made the changes (in the new branch) to create the PR. Go to your repo on GitHub, and click on "branches" (next to commits, just above your code). Find your branch in the list, and to its right, click on "New pull request".

**Your branches**

| issue-1 Updated 8 minutes ago by username | 0\|1 | 🎋 New pull request 🗑 |

Write a good description of your PR, explaining everything that has been changed in it. It's good practice to mention the issue that originally motivated the changes.

Click on "Create pull request". The PR will be created with an ID (**#2**).

Imagine now you are the project manager and click on the gear icon next to "Reviewers" to assign someone to review the PR. For this task assign yourself but of course, in practice, your project manager will be assigning a co-worker instead.

## CODE REVIEW

Because getting a PR accepted can have so many implications for the project, it's important that your code is reviewed by another developer. This is often how easily-overlooked mistakes are spotted.

Imagine now you are a code reviewer and click on "Commits" in the PR on GitHub. Here you'll see a list of commits made to the branch the PR is for. For an all-encompassing view of changes made to the project, click on "Files changed" on the same top bar. This will show all changes made to the project in a human-readable diff view. This process remains the same for most of the languages, for example Java, JavaScript, HTML, etc.

Here's an example:

```
⤵     @@ -89,6 +90,14 @@ public int compareTo(Worker worker) {
⤴
 89    90              return this.motivation - worker.motivation;
 90    91          }
 91    92
       93  +     public void read_file(String file){
       94  +         BufferedReader reader = new BufferedReader(new FileReader(file));
       95  +         String line;
       96  +         while((line = reader.nextLine()) != null){
       97  +             System.out.println(line);
       98  +         }
       99  +     }
       100 +
 92    101         @Override
 93    102         public Integer getKey() {
 94    103             return motivation;
⤵
```

As a reviewer, you might remark that the method name violates Java project standards (should be camelCased, not snake_cased), and that the BufferedReader is never closed.

If you find a mistake in your PR, hover over the offending line and click on the blue "+" button to its left. This will let you write a comment on the code. Once submitted, this will be visible to the original developer.

You as the developer can now make the requested fixes and commit and push them to have the PR updated automatically.
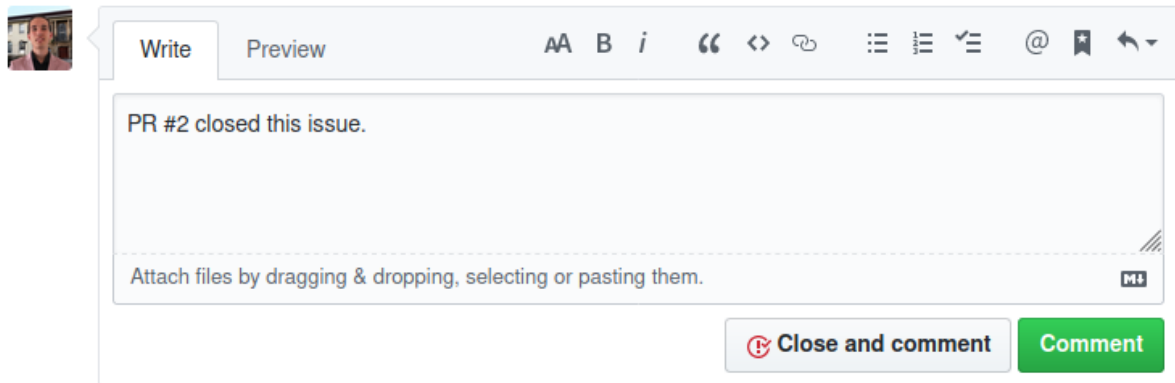
Once pushed, as the reviewer, you should go back to the "Files changed" tab and click on review changes (top right) to accept the changes. GitHub restricts review usage so that PR authors can't review their own code, so don't worry about doing this formally right now.

## FINALLY MERGING

After all this work, the branch is finally ready to be merged. Either the code reviewer or the project manager will merge the pull request. Head to your PR on GitHub, click on "Conversation" and scroll to the bottom where you should see a big green "Merge pull request" button. Note that continuous integration may pose further limits (beyond code review) before allowing a merge to happen. This may include successful building and passing of tests for your code. Setting this up is quite involved and typically not the software engineer's responsibility. You can ignore it for now.

Click "Merge pull request" and change the description if you want to. GitHub will now automatically merge your branch into master.

Now head back to the original issue, scroll to the bottom and make some closing remarks to show that the problem has been resolved. It's a good idea to mention the PR that officially closes the issue.



Click on "Close and comment" to officially mark the issue as closed.

To have the changes reflected on your computer, switch to the master branch and pull the update:

```
git checkout master
git pull origin master
```

**Take note:**

Git can be useful whether you're coding an app on your own, or if you're a company with thousands of employees. But after today's task, you should understand how it's used differently in the latter case. When programming on your own, you don't have to create issues or merge in pull requests. This is far too slow a process if you're just one person. Especially for this course, it's perfectly acceptable to complete a task without making one commit.

For capstone projects, it is recommended, however, that you make use of Git and push your work to GitHub. If you want to impress future employers it is a good idea to make use of multiple branches, issues and PRs. But this is at your discretion.

## Compulsory Task 1

- Pick any one of your GitHub repos.
- Create 2 issues for things you think could be improved. Ideas for improvements include making new methods, adding constants, renaming variables and functions, or adding comments.
- For each issue:
    - Create a branch with a meaningful name.
    - Implement the changes required by the issue.
    - Commit and push your work.
    - Create a PR for your changes.
    - Merge in the PR and close the issue.
- In a text file called **repo.txt**, paste the link to your repo. Add the file to this task's folder.

## Optional Bonus Task

Contributing to open source is a great way to help out software initiatives. It shows the developer community that you care and it shows future employers that you write quality code that's on-standard for open source projects.

Find a project you would like to contribute to on GitHub. The explore page usually has some ideas if you're browsing: **https://github.com/explore**.

- Look at the open issues for your chosen repository.
- Read through their descriptions and pick the one you think you can tackle. Some issues (especially on big projects) will be tagged with the label "Good first issue", which is a great place to start.
- Fork the repository. Help **here**. This will duplicate the repo to your account. This allows you to make changes (and eventually a PR) without altering the original project. It's standard procedure in open source.
- Clone your newly forked repo and create a new branch for the issue you're tackling.

- Most open-source projects have a "contributing.md" file that specifies how you should go about making changes to the project. Be sure to read through it before starting your coding.
- Make the required changes and test your code to make sure it works. This could take a while. Remember to commit and push along the way.
- Create a pull request from your branch to the original repo. Be descriptive and mention the issue you're resolving.

If all goes well, your PR will be accepted and you will have contributed to an open-source project. It's likely that a more familiar contributor to the project will review your code and make suggestions for changes. If this happens, be sure to implement the suggestions and commit and push them again to update your PR.

It may happen that your PR gets rejected if, for instance, the project owners don't approve of the way you tried to resolve the issue. If this happens, do not be discouraged. Pick a different issue, or a different project and try again.

Rate us
# Share your thoughts

HyperionDev strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think that the content of this task, or this course as a whole, can be improved or think we've done a good job?

**Click here** to share your thoughts anonymously.