Getting Started with Git

What you should have already received...

A1 Checklist:

Before starting on this document, you should confirm that:

1. You have access to the University's GitLab

A2/A3 Checklist:
Before starting on this document, you should confirm that:

1. You and your team members all have access to the Unit GitLab group
2. You've decided amongst yourselves who will create the repository

If any of these are not true, please email FIT1008.Clayton-x@monash.edu.

A bit about Git + Gitlab

Why Git?

You're probably used to just having a single local folder for assignments. Maybe you're a bit ahead of the curve and use Google Drive / OneDrive / Dropbox to share these documents with team members.

Git is a much more focussed application for team (or even solo) coding work, and makes working on code as a team much much easier (And becomes exponentially more useful as the size of your projects increase).

In short, Git (and by extension GitLab) allows us to:

- Itemize changes to the project
- Create a standard review process for changes
- Create a standard process for logging changes
- Run tests automatically when things are changed

Ok but what is it?

Commits

The main feature of Git is that changes to the project are separated into **Commits**. Commits are little changes to the code that can be stacked together to form the entire project. For example, the change to finish an assignment task might be split into multiple commits:

- Commit #1: Add basic documentation and file header
- Commit #2: Add an attempt at task 1
- Commit #3: Add some rudimentary tests for task 1
- Commit #4: Fix that initial attempt at task 1
- ...
- Commit #101: Finally finish task 1. On to task 2... 😢

So, rather than OneDrive, where you might only have access to the latest version of every file, we can look back and see what the project looked like a few days ago. This also splits each commit up by author, so we can see who did what.

Big deal though, many file sharing platforms allow you to see version history, so what makes Git different?

Branches / Merging

Well, let's suppose that you and a friend are both working on the assignment, and end up both making changes to a single file. You both make commits locally, and then put them on the online platform - Which one should Git take? The latest one, and override the old changes?

Fear not, for Git has a better solution - accept both changes, and allow them to co-exist!

Rather than just storing commits in a LinkedList, one commit leading to the next, we can instead store a branching structure - with each commit pointing to the change it was made on top of.

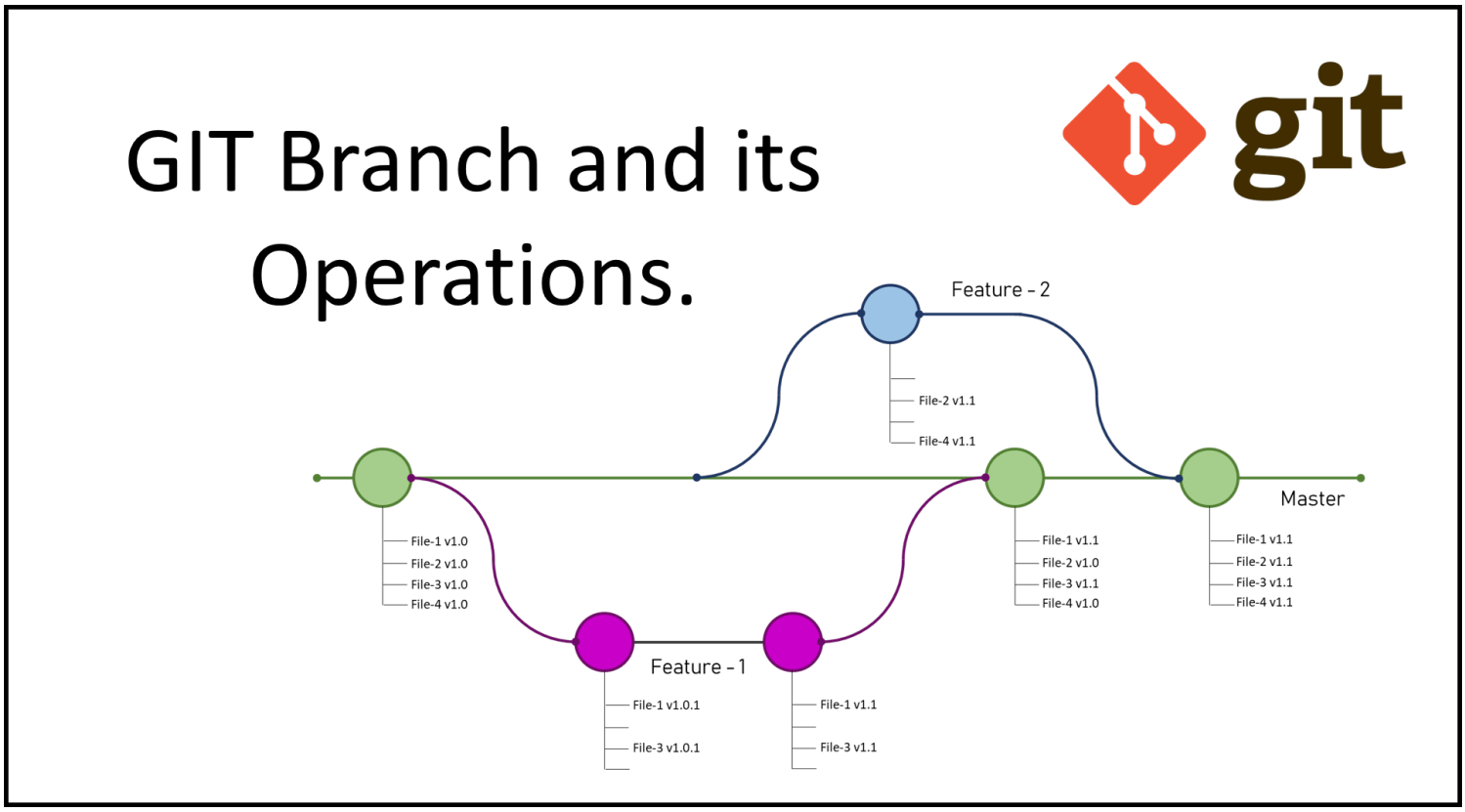


Image Credit: https://digitalvarys.com/git-branch-and-its-operations/

The green line in the image represents the "Main" or "Master" branch - the changes everyone agres on. After both group members have made their changes, they can request to merge these changes into the main branch, in a particular order. There might be some changes made to make the second merge compatible with the first.

We'll see how merging works later on, when making our own merge requests.

GitLab

Git itself is just a command line tool, for commiting work locally. **GitLab** allows us to share this work with others, and also gives us a few neat features, such as:

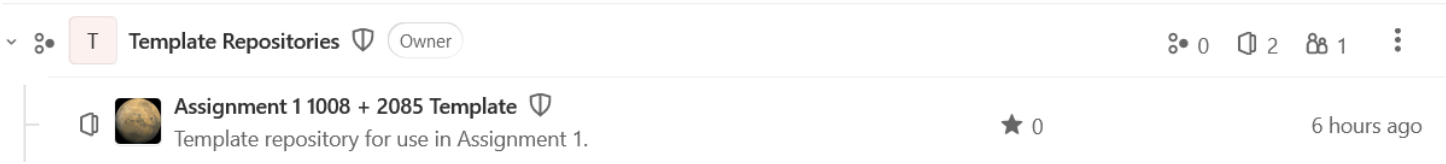
- Merge Requests, which have comments, approval process
- Running Tests when the files are changed
- Add Issues for tracking what needs to be done
- much, much more.

We'll see most of this in the following pages.

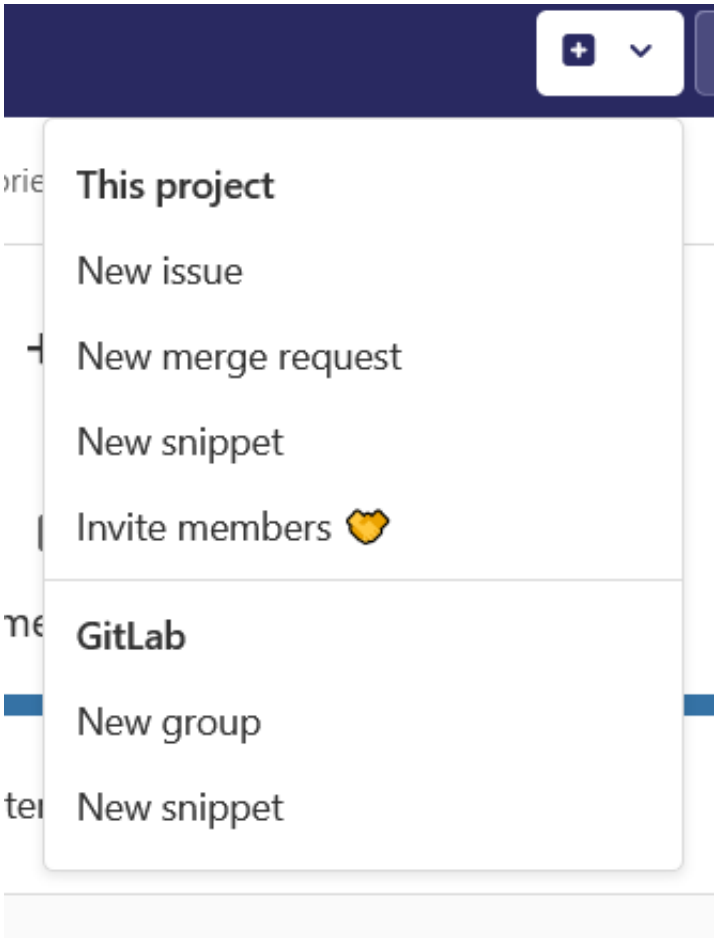
Getting Started - Creating your assignment repository

Before we get started with Git, let's first make sure we can create a repository for our assessment on the FIT Gitlab.

Start by going to the 1008 Group on Gitlab. You should see the "Template Repositories" Subgroup:



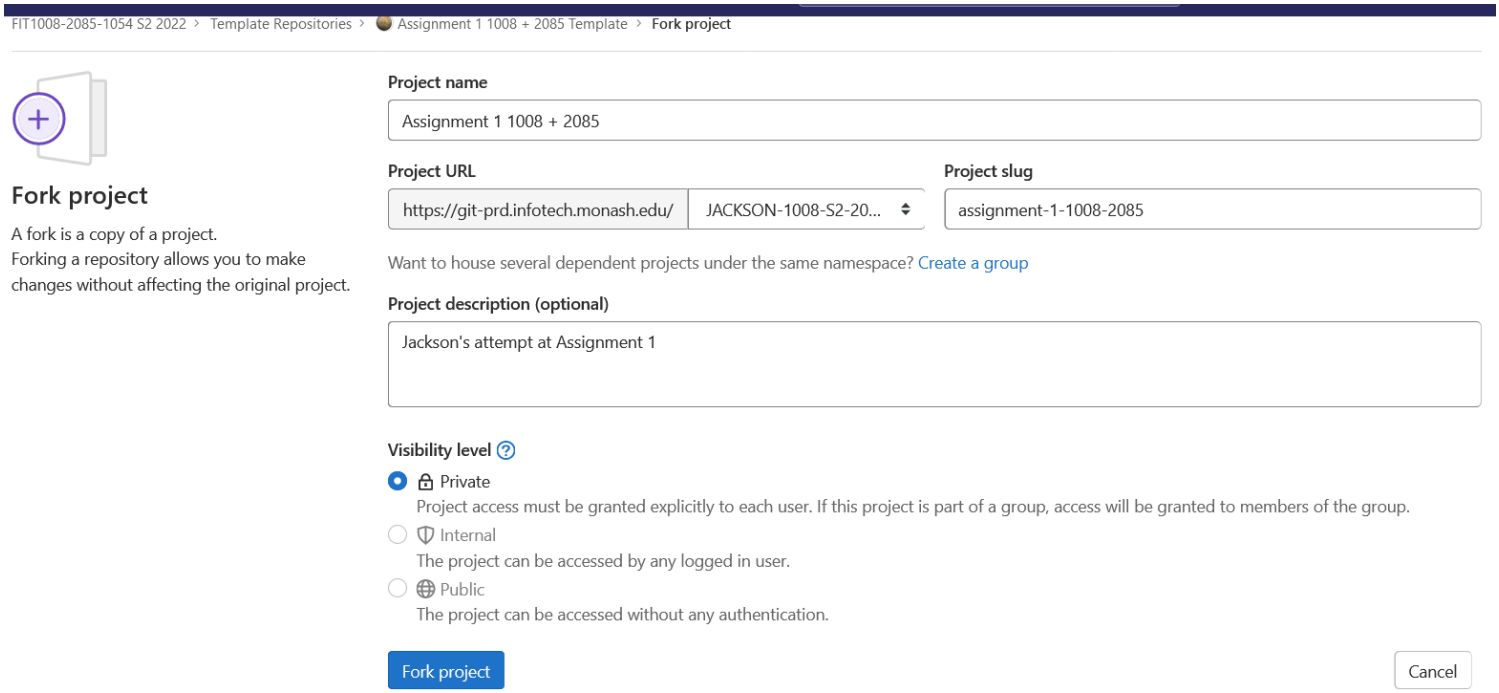
Click on the Assignment 1 Template associated to your unit code to take you to the landing page. Here you can see the template files given for the associated assignment. We are going to fork this template into own own group. We'll start by creating our own group. You can do this by clicking the little `+` icon at the top of the screen and selecting "New Group"



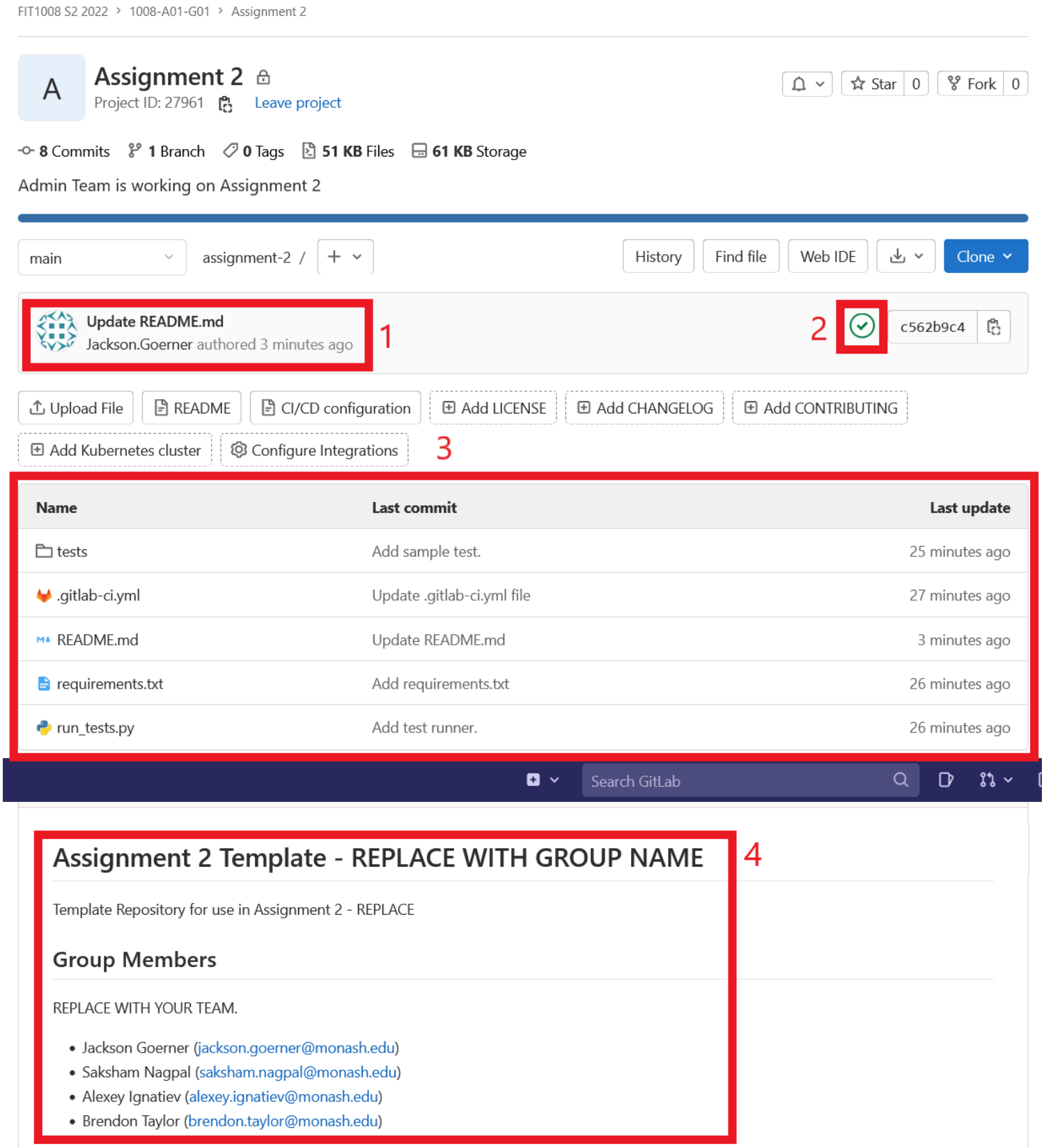
From here, select "Create Group", give it reasonable name like "1008-<STUDENT_ID>-ASSIGNMENTS-S2-2022", and make sure to **keep the visibility settings to Private**. Once this group has been made, navigate back to the template repository earlier. We are going to *fork* this repository into our own group.

To fork, we simply need to press the little "Fork" button at the top-right of the repository. When

forking, set the namespace to the name of the group you just created, and set the project name to something reasonable.



This should create your own copy of the repository in you own group, and should look something like this:

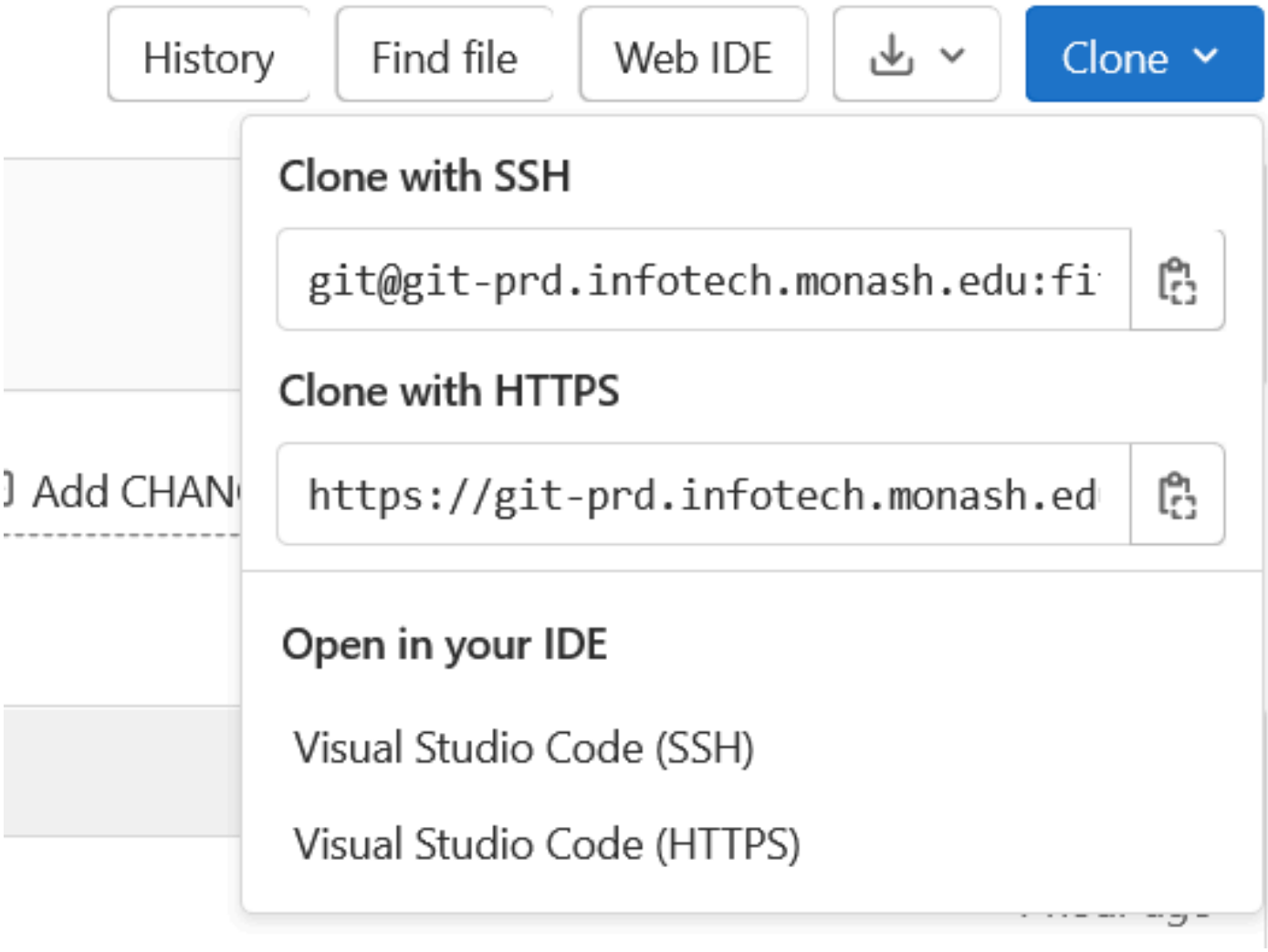


1. This is the **Latest Commit** to the project, so you can see the latest changes
2. This is the result of any **CI/CD** checks that are part of the project. The template repository has a few basic test cases, which you can see the result of by clicking on the tick, then the "Tests" tab.
3. This is the contents of the repository, where your submission will be marked. The template probably has different documents to what is shown in the picture, but some things will be the same.
4. The project long description. This pretty document is generated based on `README.md`. We'll learn how to modify this document later on.

Installing Github Desktop, Setting up a Personal Access Token and Cloning the Repository

For this tutorial I'll be using Github Desktop, since I've found it the easiest to use. If you are already familiar with git and have another system, feel free to use that, but if you are new to this I'd recommend following the instructions given.

After installing and following the prompts, you should be able to go to "File > Clone Repository". At this point, you need to get the repository URL. To do this, navigate to your assignment repository (the one in your group, not the template one), and click "Clone", and get the HTTPS link (second one in the dropdown)

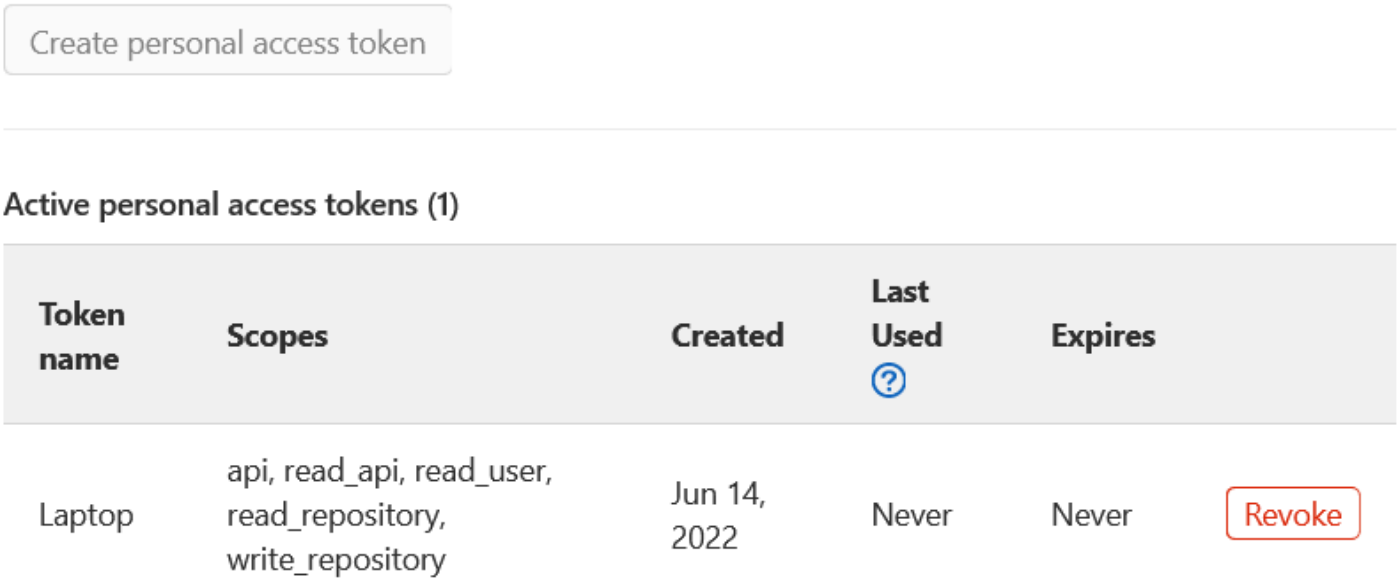


Copy this into the Github Desktop window, select where you want to clone the repository to on your computer and... You get an authentication error.

No worries! To fix this, we need to add a Personal Access Token onto our account. To do this, in GitLab click on your profile picture at the top right and select "Edit Profile". On the left sidebar, select "Access Tokens".

From here we can name an access token, and give it specific privileges (If you are unsure, just tick all boxes), and then click "Create Personal Access Token". This should give you a token at the top of the page. Copy it and Save it somewhere.

The bottom of the page should now look like this:



Now, back to the Github Desktop authentication. The username for authentication should be your university email, and the password should be that personal access token we just created.

Finally, let's open the newly cloned repository in your IDE of choice (VSCode, Pycharm, etc).

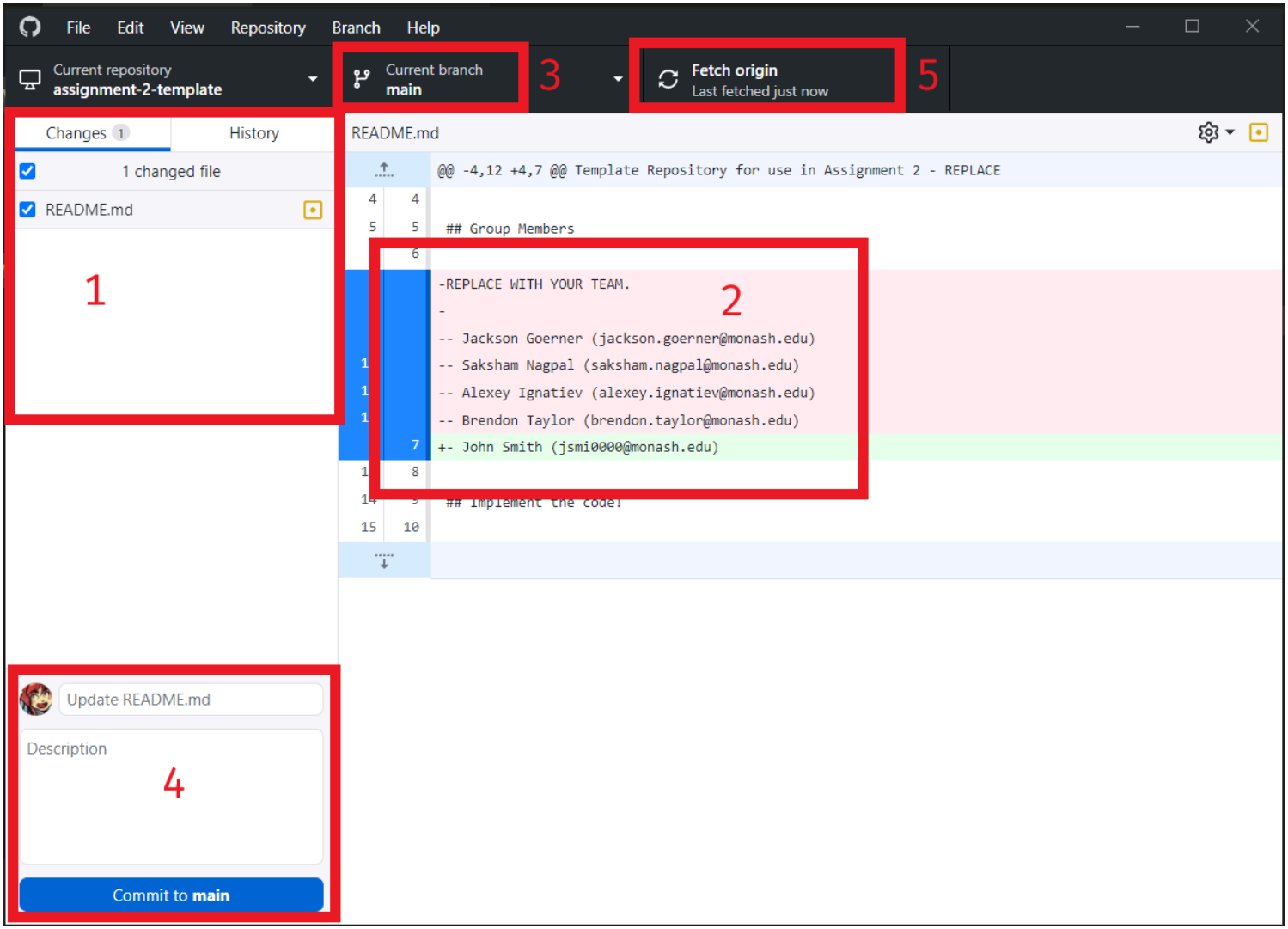
> Press the "Open in File Explorer" button if you forgot where it installed to.

Using Git: Commits, Branches and Merge Requests

The Main View

Now that we have the repository locally available, we can start making changes. Let's begin by removing the admin team from README.md and adding ourselves.

Open README.md, edit the file, and save. Opening Github Desktop again, your view should be very different:



Let's discuss everything seen

1. This is a list view of all files that have been changed. The checkbox next to a file defines whether a particular file will be included in the next *Commit*.
2. This shows a detailed view of a selected file. Red, prefixed with `-`, denotes what has been removed, and Green, prefixed with `+`, denotes what has been added.
3. This box tells us we are currently on the **main** branch. to change branch we can click this box.
4. This box is for creating *commits* based on our changes. A commit has a name and short

description, which should be edited here.
5. Because there might be differences between your local repository and the online repository (origin), this button shows if we are up to date or not. You can refresh and pull changes by clicking this button.

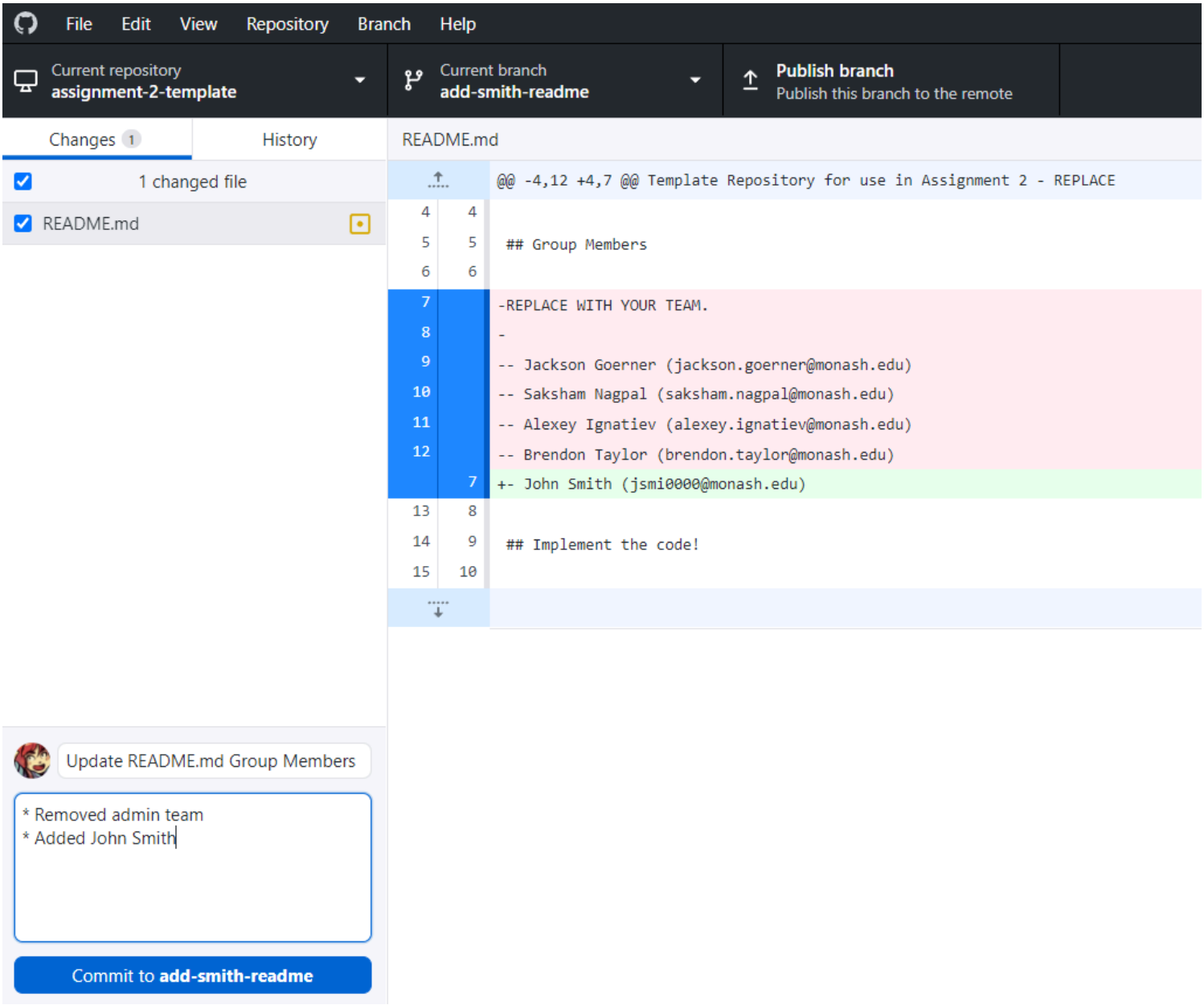
Ok, but before we commit directly to main, let's try make this change in a way that won't disrupt our team members, by making a **New Branch and Merge Request**.

Commiting directly to main can make it very hard to collaborate as a team, **especially** if you are not constantly pushing changes to origin. To avoid this, and allow collaboration on the same file, users make changes on separate *branches* and merge this together

Episode II: A New Branch

First off, we should break off from the main branch. To do this, click box number 3, and press the "New Branch" button. Give it a reasonable name, like `add-smith-readme`, and press "Bring my changes to add-smith-readme", so that your changes to `README.md` are copied across to the new branch.

Finally, give your commit a name and short description. Github Desktop should now look like this:



You might notice a few things, namely:

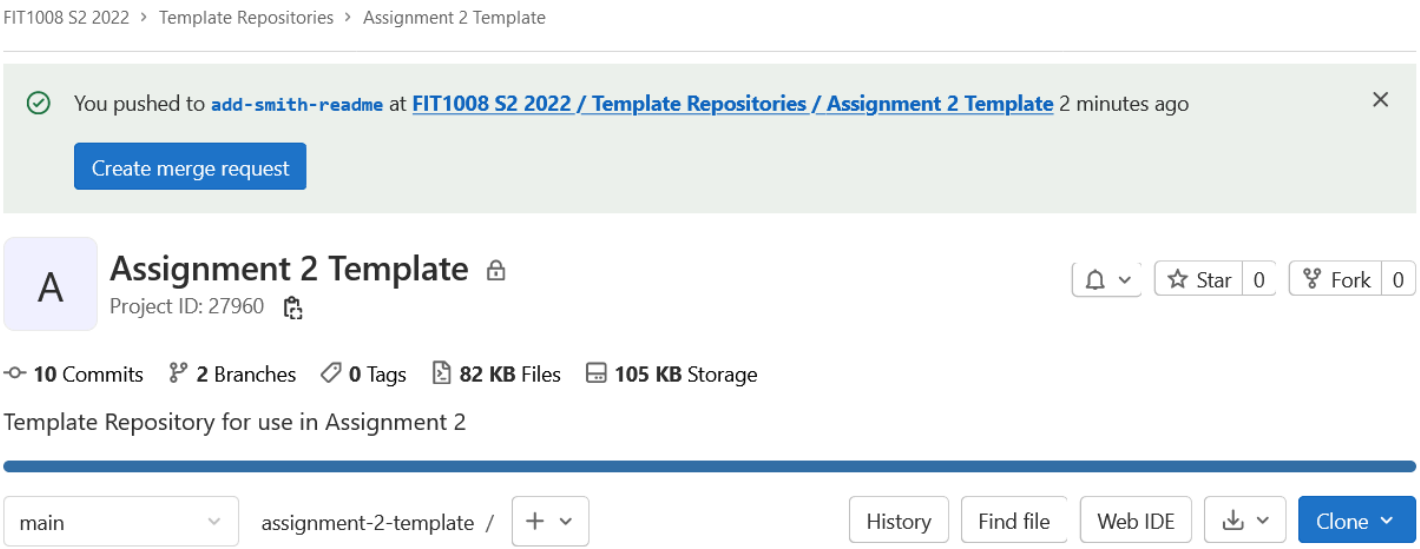
- The commit box message has changed from "Commit to **main**" to "Commit to **add-smith-readme**"
- The branch box has changed
- The origin box has changed from "Fetch" to "Publish". This is because currently **add-smith-readme** is *not on the online repository*.

So, let's press the big blue button to commit, and then press "Publish" to push these changes to the online repository.

It's generally good advice to commit regularly after every small change, even if you don't give the most informative commit messages. This makes it easy to track the order in which things were changed, and how these changes built up.

Remember after to push online every commit, or you might lose your work!

Now, refreshing the Project page, you will hopefully see:



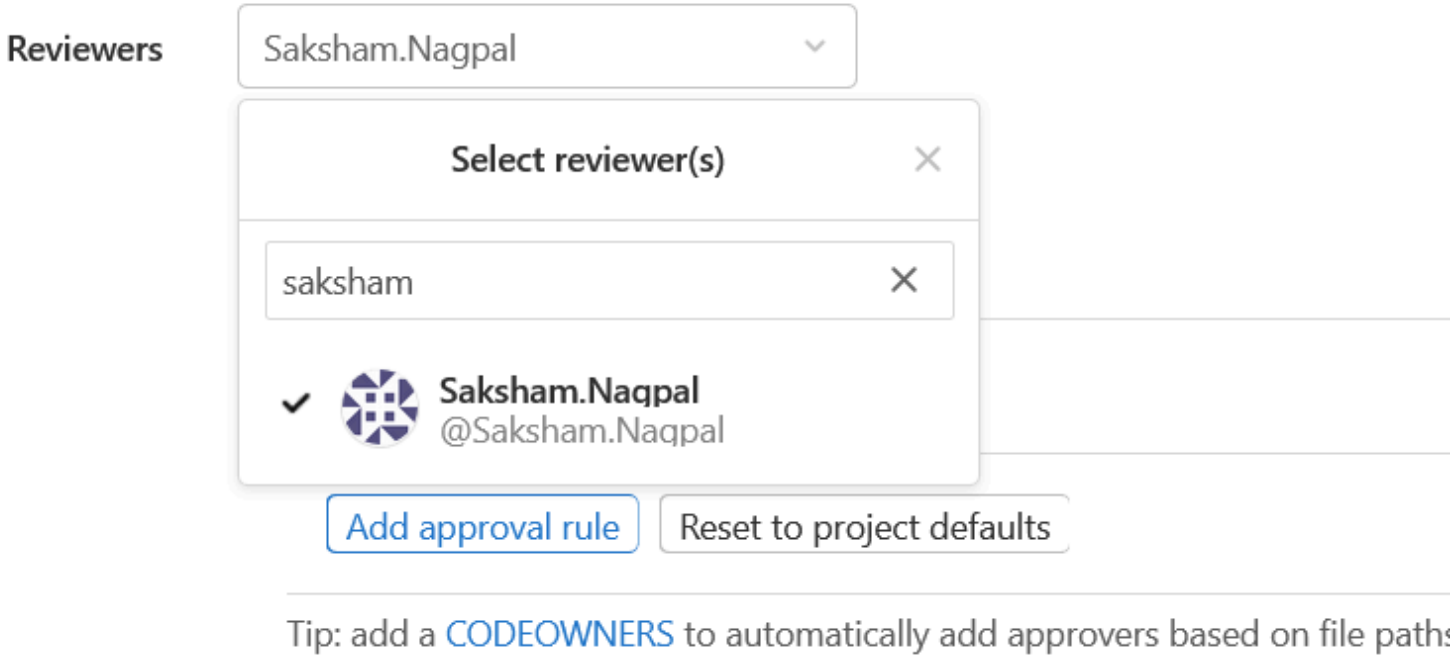
You can press the "2 Branches" button to view all existing branches and make merge requests that way if the notification doesn't show up.

Yay!

Merge

Let's create a merge request, and get a teammate to review.

You'll notice that most of the information is added for us, based on the commit message and description,

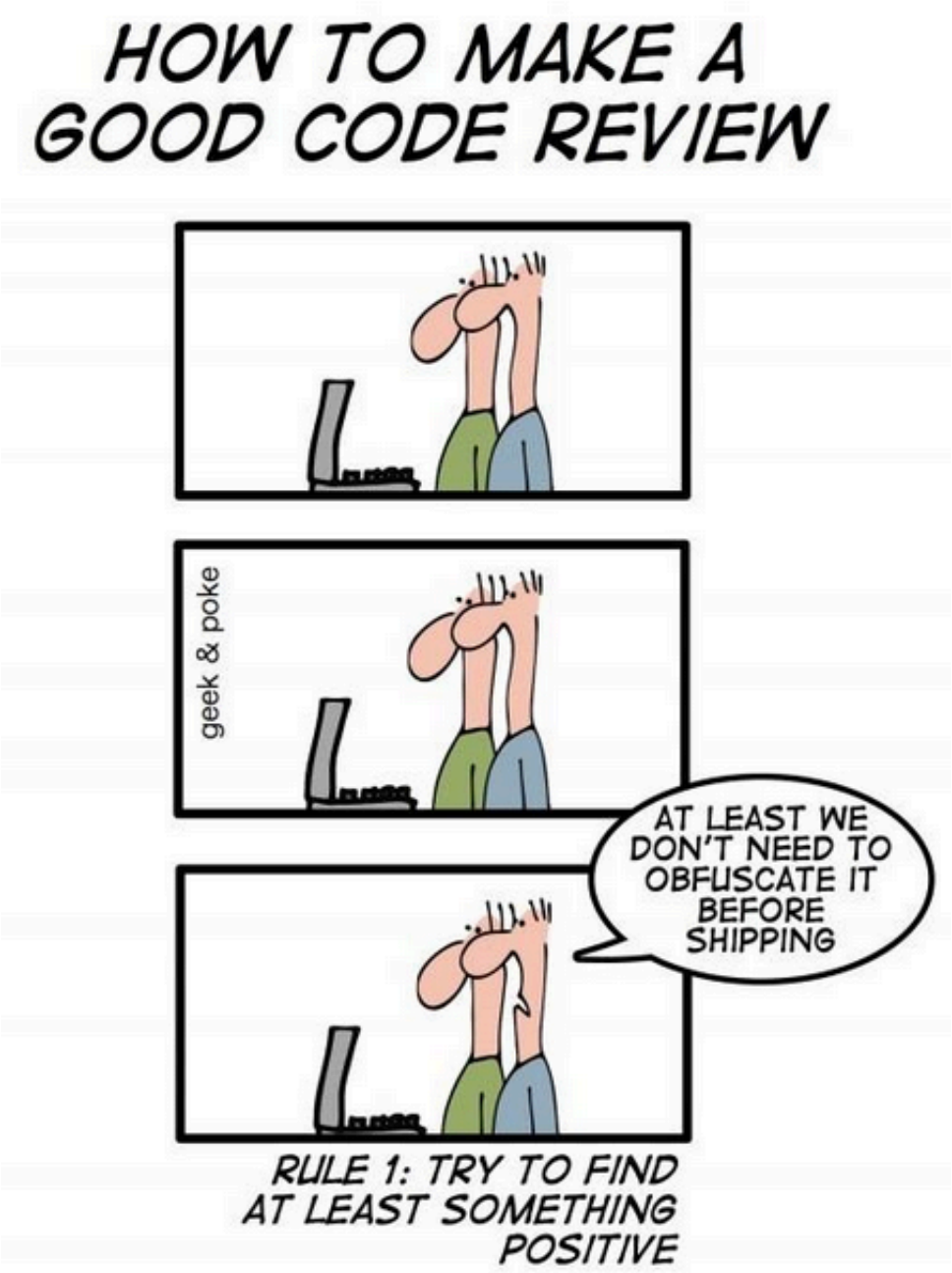


Having another team member review your code before merging drastically reduces the number of errors you'll make and can save a lot of time down the track.

> There are more options here, such as squishing commits on merge, that you can explore at your own pace.

Once you are all set, press "Create Merge Request", and your teammate can review!

Review

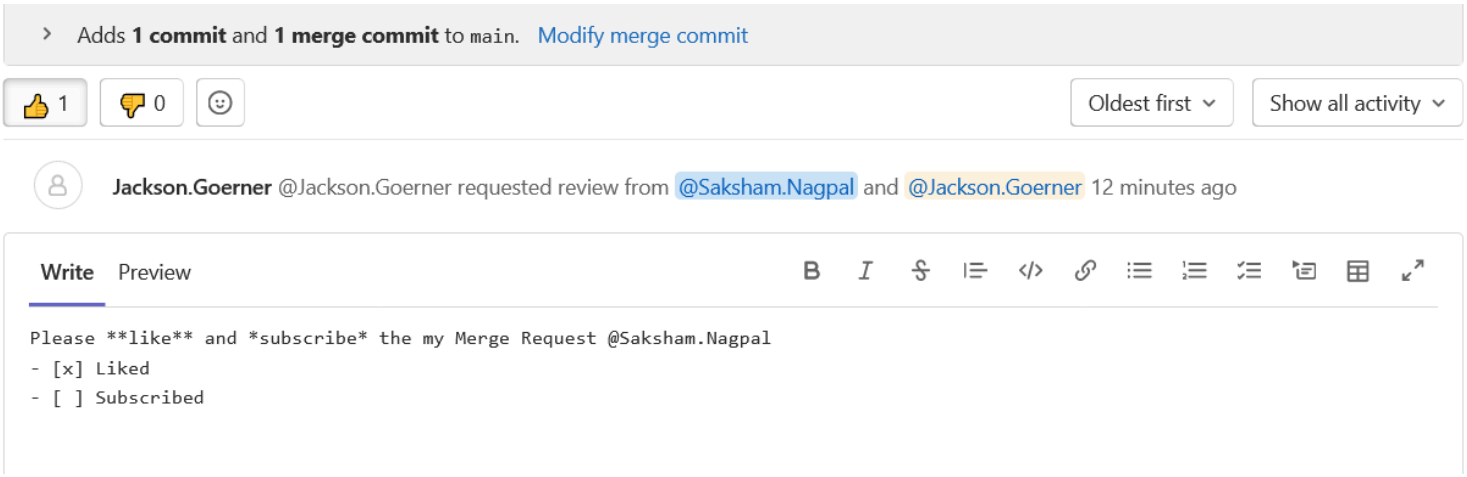


Reviewing a merge request is rather simple, and boasts many features to assist Code Review.

I'll quickly go through some things you might want to do in the Review process:

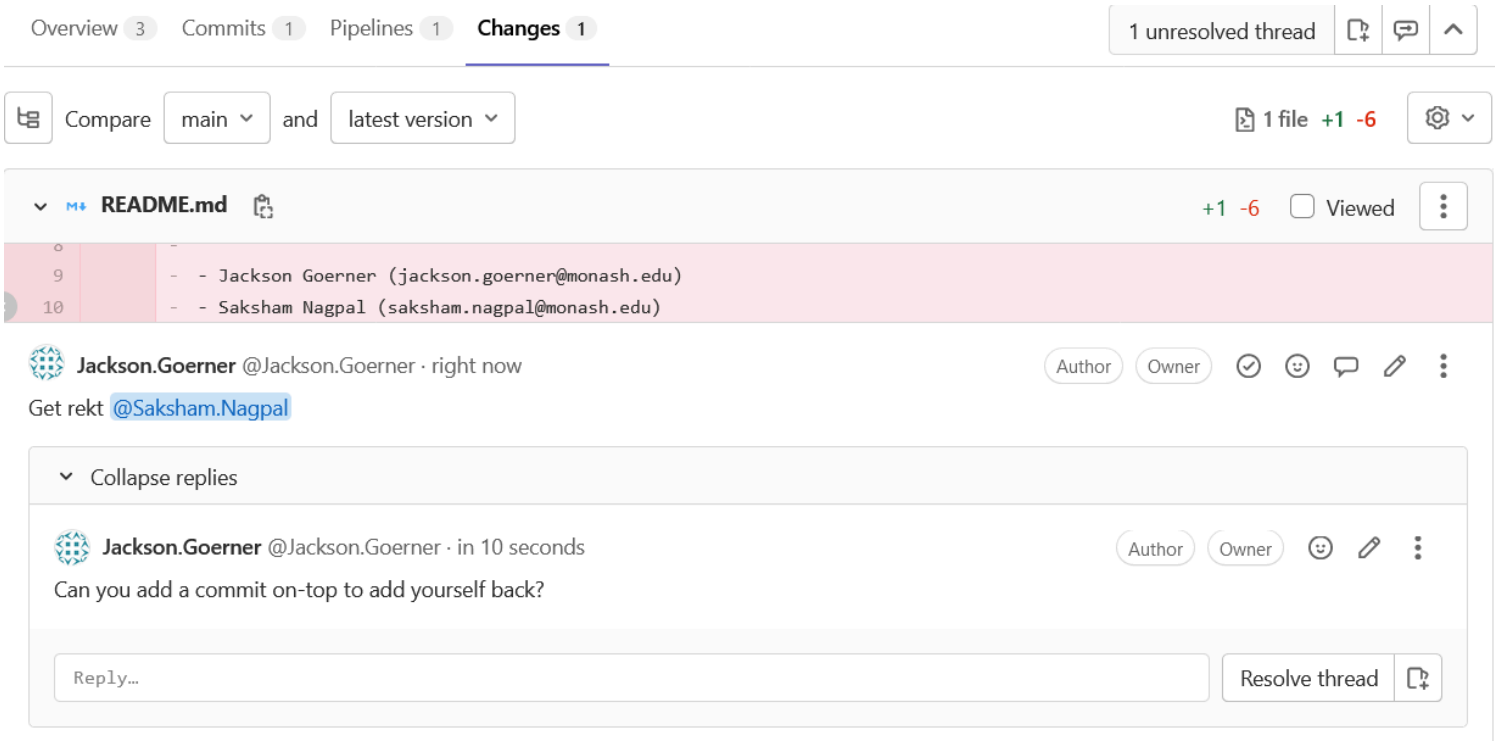
Leave a General Comment

You can leave a comment on the merge request by scrolling all the way to the bottom of the overview tab:



Review the changes and leave specific comments

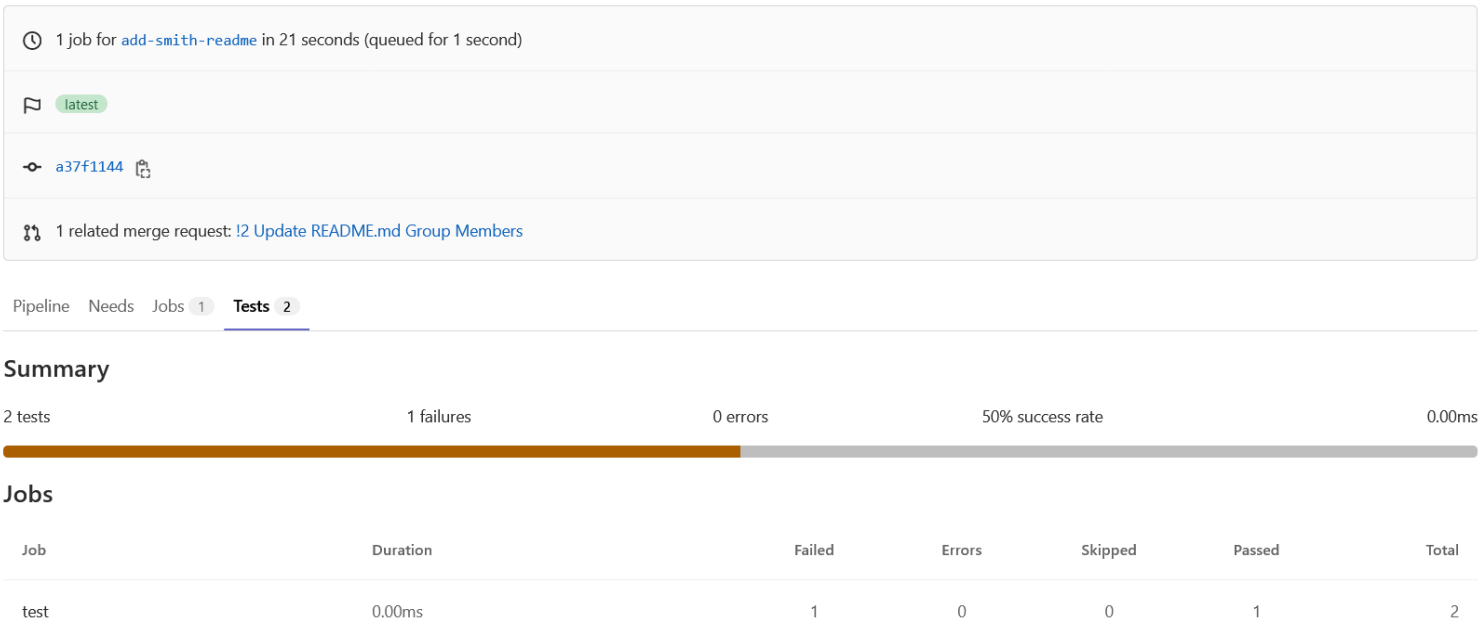
Click on the "Changes" Tab to view what files have actually changed, and hover over a particular line to leave a comment:



You can also resolve comments that have been fixed in later changes.

Check Tests are passing

On the Pipelines tab, you can see what CI/CD has been run and the results. Just click the green tick under "Status", and then navigate to the "Tests" tab.



Approve

Finally, if everything looks good, you can approve the merge request, and Merge the result into `main` !

Adding Team Members for Group Assignments

To add team members for group assignments, navigate to the page for your repository, and open the left sidebar. Hover over "Project Information" and click "Members".

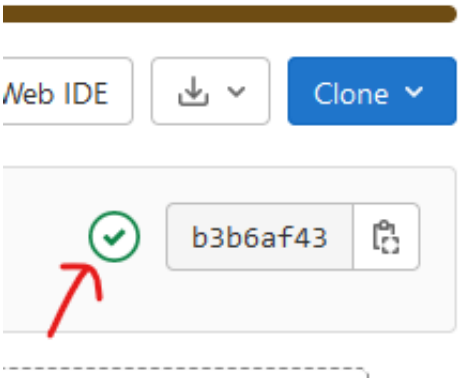
From here, you can click "Invite Members" on the top right and get your group in. Add each students email and give them the "Maintainer" role.

CI/CD

Finally, we can run tests when changes are made on GitLab. To change how these tests are run, or add more things to auto-run, you can change `.gitlab-ci.yml`.

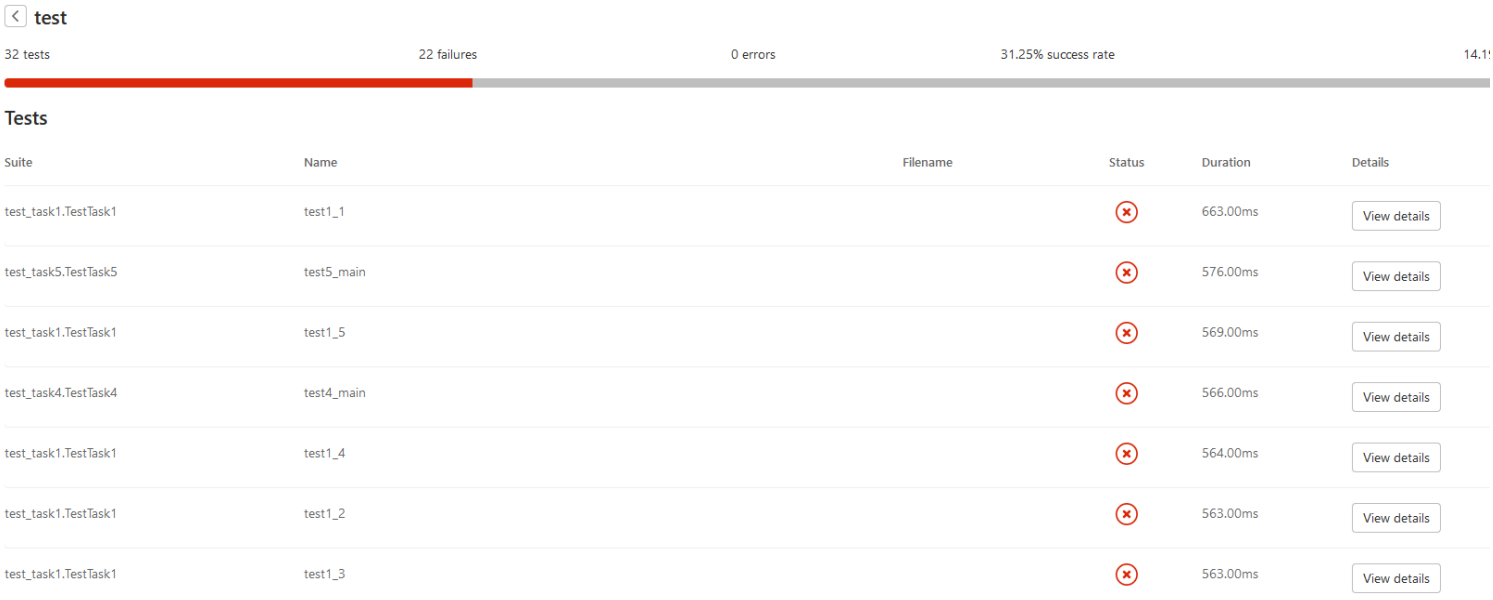
> Changing this file will not change how your assignment will be marked - So make sure your code passes the tests without changing this file, or you risk getting a 0 for all tests.

To view the tests running after a particular commit, try clicking the green tick located on the front page of your project:



This takes you to any actions that were run as part of the commit. One of these was the tests located in the `tests` folder of your repository. Writing your own unit tests for A1 is not a requirement, so we do not go into too much detail on how these work, but if you are interested, you can peruse the files `tests/base_test.py` and `run_tests.py`, and the testing file for a particular task, like `tests/test_task1.py`.

To view the results of the tests, check the "Tests" tab and click the "test" Job. This should list all tests undertaken as part of the action:



As you can see, most tests are failing (Makes you wonder why there was a green tick in the first place?). To see why each test is failing, you can click the "View Details" button.

This should show a diff(erence) of what is expected vs. what was received:

```
- You haven't started task 1 yet
+ Welcome to the Thor Electrical Company!
+ Enter your total consumption in kWh: Mr Loki Laufeyson, your
electricity bill is $138.60
```

If you get another error telling you to set self.maxDiff = None, you can do so by editing `tests/base_test.py` and adding `self.maxDiff = None` between lines 43 and 44, right before `method()`.

So our output is missing two lines, and provided one that is not needed. Notice that user input and newlines stemming from user input are not present in this comparison.