

THE UNIVERSITY OF
AUCKLAND
Te Whare Wānanga o Tāmaki Makaurau
NEW ZEALAND

Git Workshop

ELECTENG 209: Analogue & Digital Design

Getting Started with GitHub Desktop

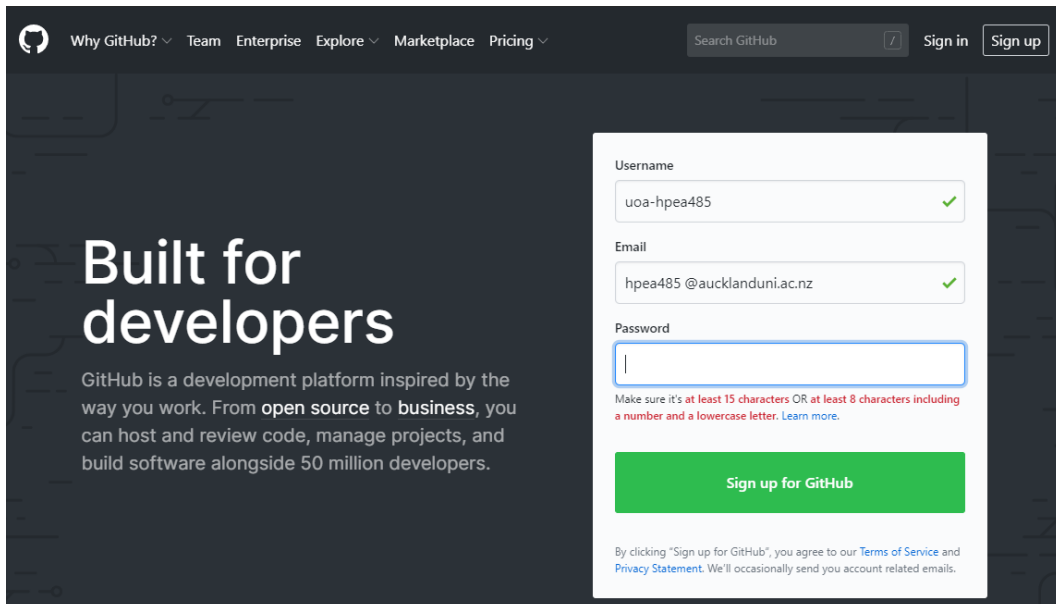
Hammond A Pearce & Duleepa J Thrimawithana

Department of Electrical, Computer, & Software Engineering

Before You Start

Create a GitHub Account

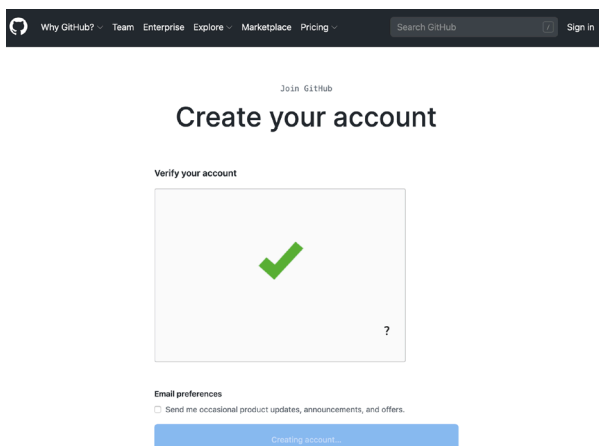
As shown in Fig. 1(a), on a web browser go to <https://github.com>, and enter a “Username”, “Email” and a “Password” to sign up for GitHub. You should use your UoA email address as “Email” to unlock educational benefits. Ideally the “Username” should be in the form uoa-UPI, where UPI is your UoA username (e.g. uoa-hpea485 for Hammond). This is because you will use GitHub throughout your UG degree and it will make it easier to identify if your details match what’s on Canvas. Alternatively, a more personalized username may be used. Note that you’ll likely continue using your GitHub account after you finish your UG studies as many companies have their employees using GitHub. Click “Sign up for GitHub” to create an account.



The screenshot shows the GitHub homepage with a sign-up overlay. The overlay contains the following fields and text:

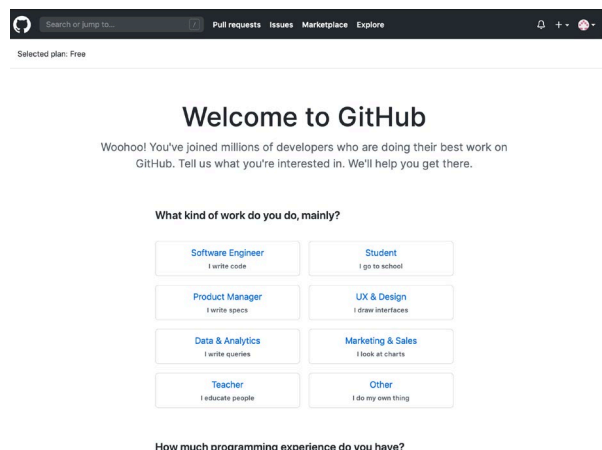
- Username:** uoa-hpea485 (with a green checkmark)
- Email:** hpea485@aucklanduni.ac.nz (with a green checkmark)
- Password:** (empty field with a red error message: "Make sure it's at least 15 characters OR at least 8 characters including a number and a lowercase letter. Learn more.")
- Sign up for GitHub:** (green button)
- By clicking "Sign up for GitHub", you agree to our [Terms of Service](#) and [Privacy Statement](#). We'll occasionally send you account related emails.

Fig. 1(a): GitHub.com



The screenshot shows the "Create your account" page. It includes a "Verify your account" section with a large green checkmark and a question mark. Below this is an "Email preferences" section with a checkbox for "Send me occasional product updates, announcements, and offers." and a blue "Creating account..." button.

Fig. 1(b): Verifying free account



The screenshot shows the "Welcome to GitHub" page. It includes a "What kind of work do you do, mainly?" section with a grid of buttons for different roles: Software Engineer, Student, Product Manager, UX & Design, Data & Analytics, Marketing & Sales, Teacher, and Other. Below this is a "How much programming experience do you have?" section.

Fig. 1(c): Providing usage information

You will be asked to verify your account as shown in Fig. 1(b). Though GitHub offers paid services, for your UG studies, the free version has all the features needed. Once you complete this step, it will ask you to provide some extra information as shown in Fig. 1(c). Then an email will be sent to verify your email address. Go to your inbox and verify your email address. This will open a new browser window asking what you would like to do first. You can skip this by clicking “Skip this for now” found at the bottom of the page as shown in Fig. 2(a). You will now be taken to your GitHub home screen shown in Fig. 2(b).

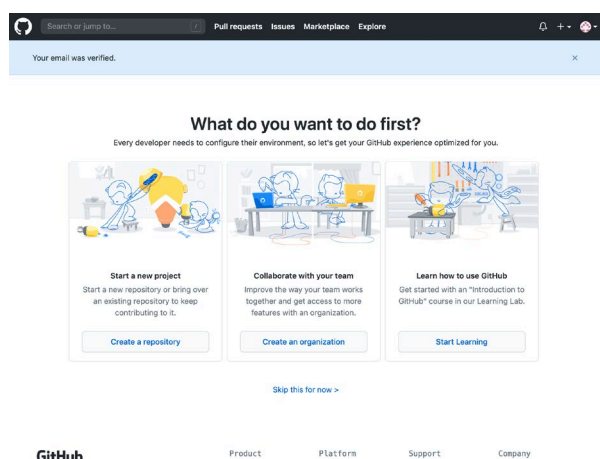


Fig. 2(a): Verifying account

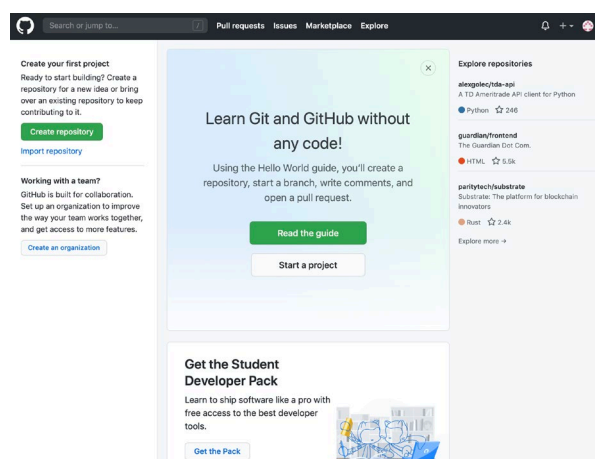


Fig. 2(b): Your GitHub home screen

While you are on your GitHub home screen, you may click on “Get the Pack” to upgrade your account and get access to many benefits for free including a GitHub Pro account. You will only be able to upgrade if you had used your UoA email address. You will be asked to provide extra information so that they can verify you are a student at UoA.

Download and Install GitHub Desktop

The MDLS computers should have GitHub Desktop installed. If you are using your own devices, follow the “[GitHub Desktop Install Guide](#)” that is available on Canvas to install and setup GitHub Desktop.

Creating a New Repository

Let’s create a new repository to learn how to use Git and GitHub **using the web interface** provided through github.com. A repository can be thought as the root folder that belongs to a specific project/activity. So, as you do with a typical project/activity folder on your computer, you can put all files and folders that belongs to that project/activity into a repository. Usually you will create a repository for each project or activity your work on. In this course, we will use 2 main repositories, one for storing all files related to the project and another to store all files related to lab assignments. As shown in Fig. 3(a), go to github.com and sign in to create a new repository from the top right menu on your GitHub homepage.

Once you click on “New Repository”, you will be taken to the create repository page, shown in Fig. 3(b). Here we have to setup following:

- Unless you are a member of multiple GitHub organisations, by default the repository owner should be set to the username of the account you created. Since we have created a new GitHub account with the username “ee209uoa”, in Fig. 3(b), the owner is set to “ee209uoa”. If this is not set to the username you created, from the dropdown menu select the correct username.

- Give a name to the repository. In this example our repository is named “my-git-tutorial”.
- It is also good to add a description to tell a bit more about your repository. However, in this case we will leave it blank.
- Ensure that the repository is set to “Private”. This means that only you, and people you choose, can access the repository. If you set it to “Public” then it will appear on public internet searches and anyone could access it. So, it is **VERY IMPORTANT** to set this to “Private”
- Make sure you tick the checkbox for “Initialize this repository with a README”. This means GitHub will create a README.md file for us and store it in our repository.
- The “Add gitignore” and the “Add license” fields can be set for “None” for this example.

Once you’ve done all of that, press the “Create repository” button. This will now take you to your repository homepage shown in Fig. 4 with several key areas highlighted.

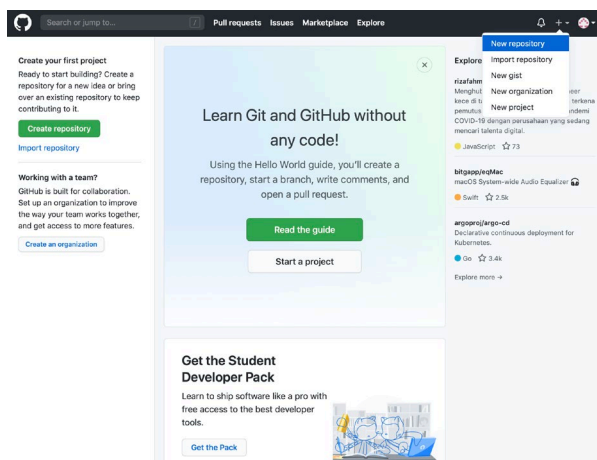


Fig. 3(a): Creating new repository

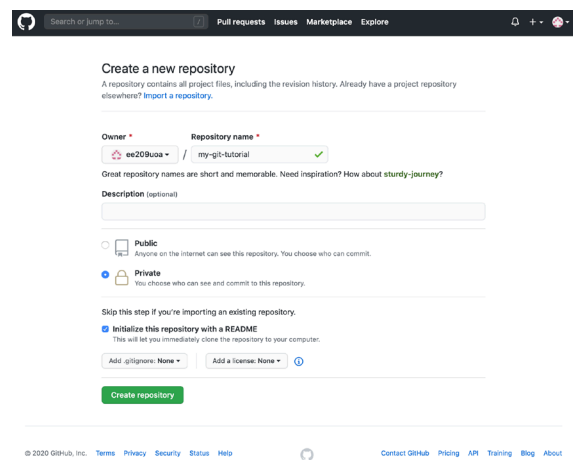


Fig. 3(b): Settings for new repository

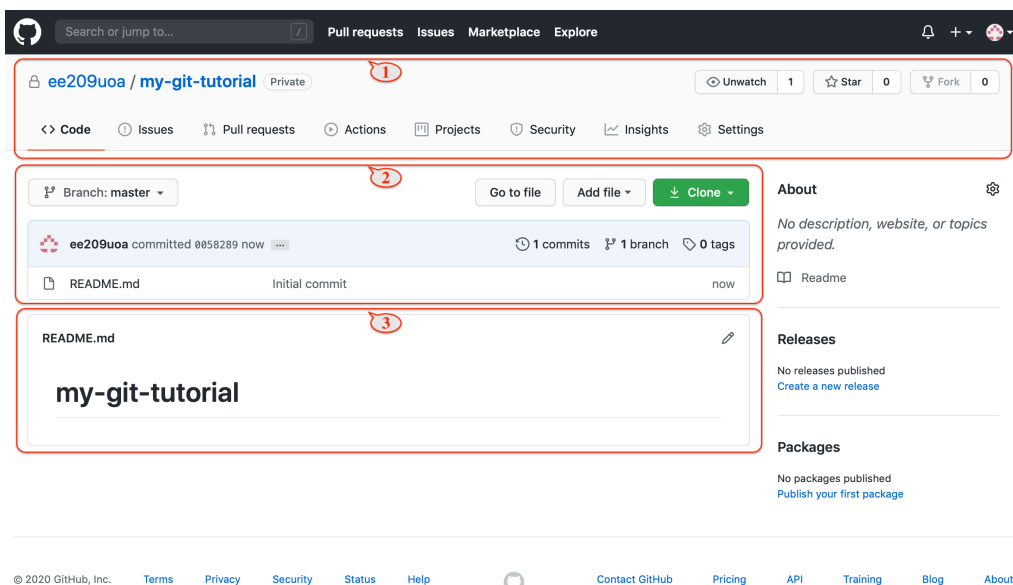


Fig. 4: Homepage of “my-git-tutorial” repository

The top area labelled “1” contains the name of the repository as well as a menu bar with all the different settings and features built into GitHub (e.g. the issue tracker, pull-requests, repository settings, etc).

The middle area labelled “2” is a rudimentary file viewer, somewhat like the “Windows File Explorer” you use every day. The “Add file” button allows you to create new files or upload new files. The “Clone” button allows you to download (also referred to as clone) the repository to your computer.

The bottom area labelled “3” is a viewer which will show the contents of a file that is either named README.md or README.txt. As you can probably intuit from this feature, it has always been strongly encouraged in software development to create explanatory READMEs and put them in your software source folders. Since we have already created a README.md, it shows the contents of this file in the area labelled “3”. By default, if you select “Initialize this repository with a README” when you create a new repository, GitHub adds the repository name and description you have provided to the README.md it creates. Hence the reason you see the repository name in the README.md file.

Note that a file with .md extension is a Markdown file while a file with .txt extension is a text file. Oppose to a text file, Markdown files allow many features and you can explore some of these features [here](#).

Editing on GitHub

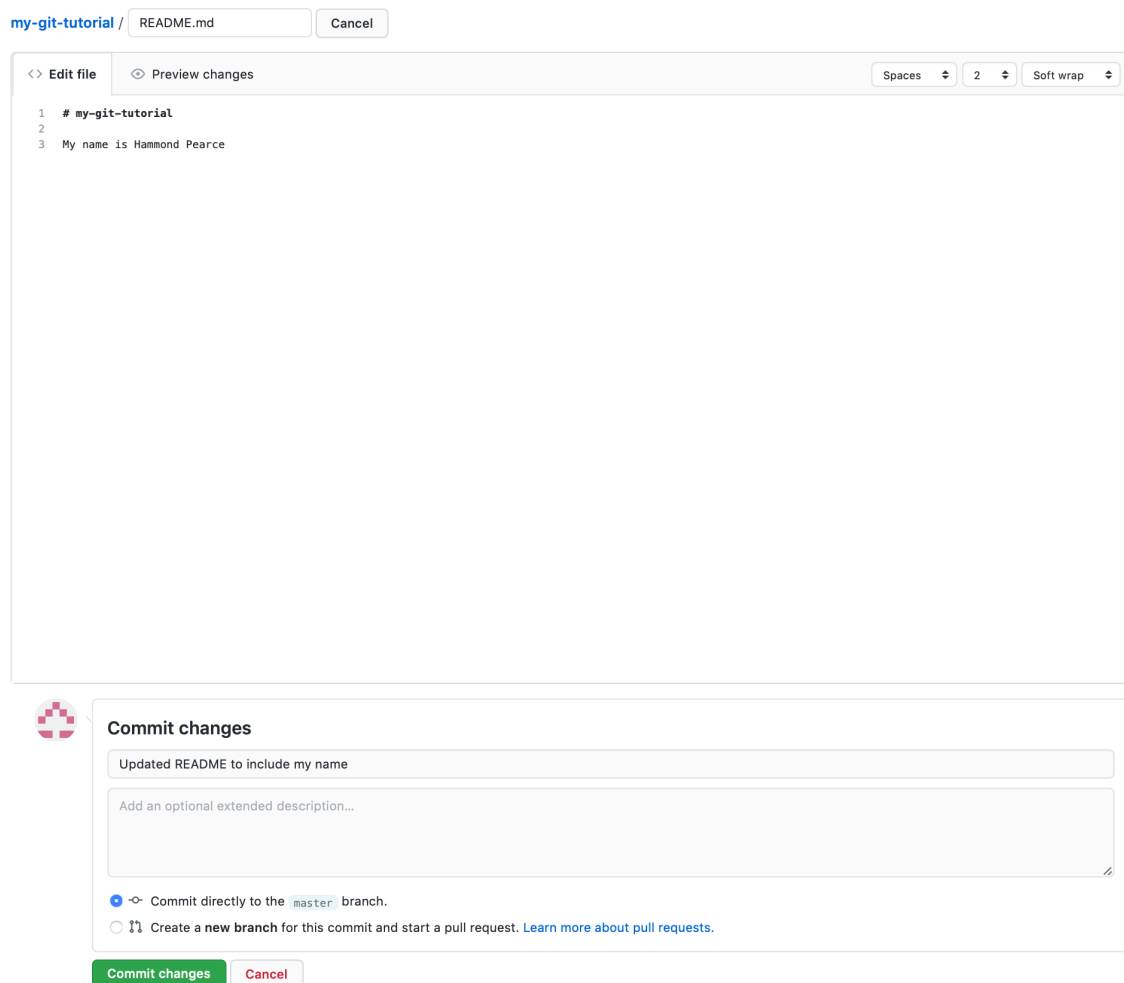


Fig. 5: Editing pane

Let’s edit the README.md file on GitHub through its web interface. You wouldn’t normally edit files directly in GitHub, but the web interface supports editing markdown (.md) and text files directly. To edit the

README.md file, press the pencil icon at the top right of area labelled 3 in Fig. 4. You will be taken to an in-browser editing pane. In here change the file by adding your name as shown in Fig. 5. Once you’ve done this, add a commit message to say what you did in the text box provided just below “Commit changes”. In this example we have added “Updated README to include my name” as the commit message. If you have done many changes it is also good to add a descriptive message below this so that team members working on the same project know about the changes you have done. Click the “Commit changes” button to commit (think of as saving) the changes you made to the file.

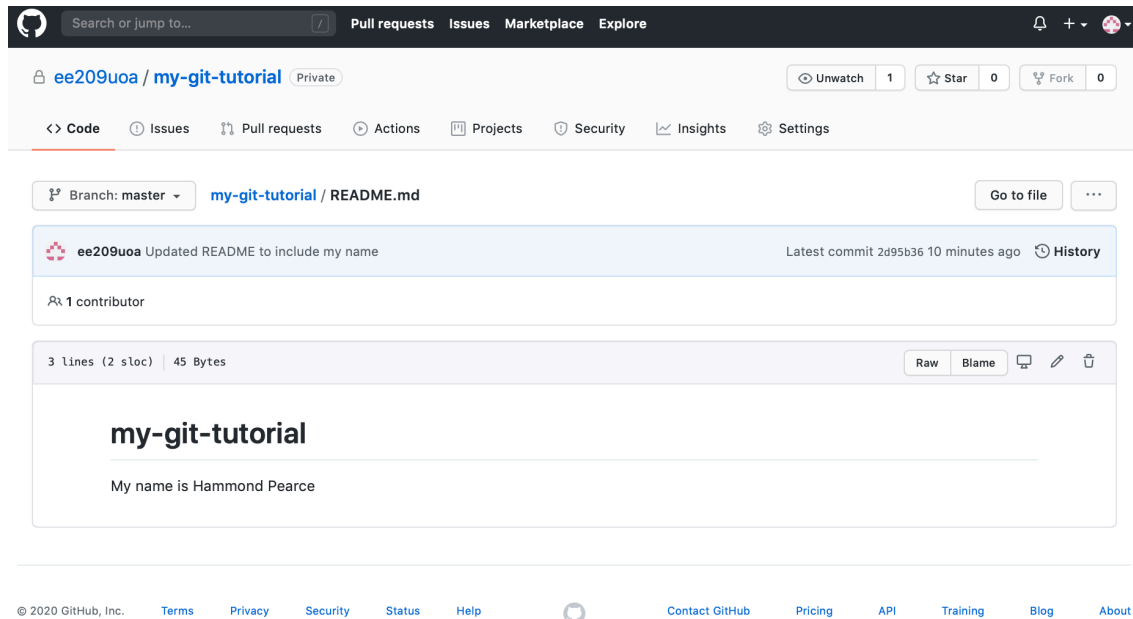


Fig. 6: README.md file view

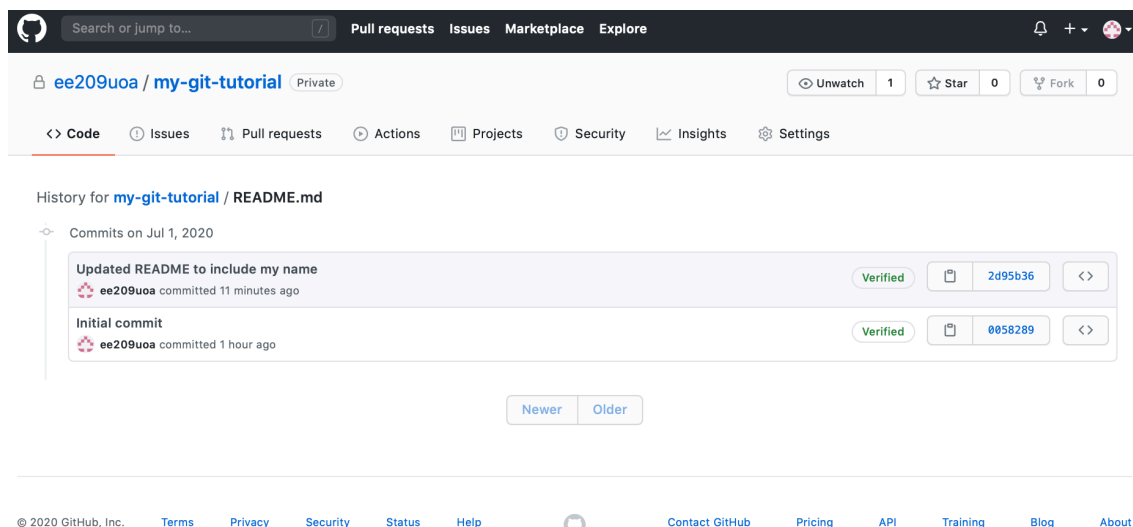


Fig. 7: History of commits made

You’ll now be taken to the file viewer in GitHub, which will show you this file with your changes as in Fig. 6. If you press the “History” button you will see the changelog of the file. You can use this viewer to inspect previous versions of the file. As you can see from Fig. 7, each change is identified by the commit message provided at the time of commit. Therefore, it is very important to provide meaningful commit messages.

Let's return to the repository homepage by pressing the link "my-git-tutorial" at the top of the page next to your username. In Fig. 7, it is shown as "ee209uoa/my-git-tutorial". On the home page, shown by Fig. 8, you'll see the updated version of the README.md file again, as well as the file viewer as before. Now in the file viewer you can also see your most recent commit message.

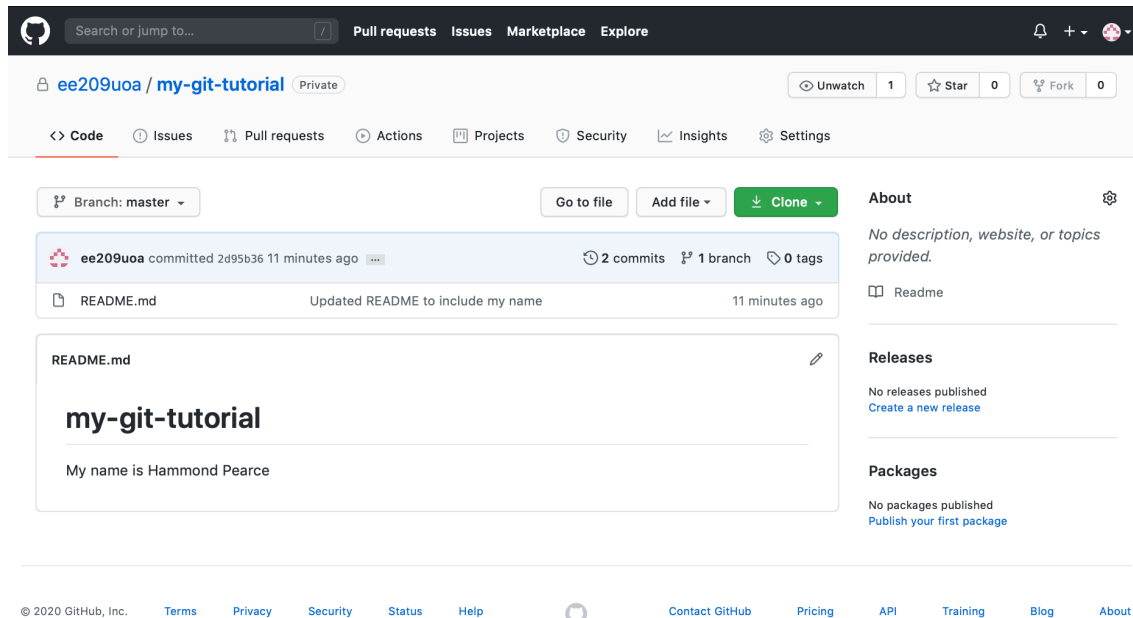


Fig. 8: Repository homepage showing changes made

Let's take a quick recap. A **repository** is like the root folder. You can put files and folders that belong to a single project/task in it and save them. Importantly, **repositories** let you save **every version** of a file, not just the most recent. You do this through **commits**. **Commits** are like **snapshots**. They are a picture of the **complete repository** at the time the commit was made. When we updated and committed the README.md just now, we saved the file and took a snapshot of it, and the repository now has two versions saved. Let's now look at how we can work with a repository on our local computer.

Cloning

GitHub is a service provider for Git, which is an open source version control framework. There are a number of providers for Git, including GitHub, Bitbucket, and Gitlab. There are also a number of desktop clients, including the open source tools provided on <https://git-scm.org> which include a command line Git interface (Git Bash) and a basic GUI. However, in this workshop we will use GitHub's own GUI, known as GitHub Desktop as this is more modern.

In the future, you may choose to learn to use Git on the command line. Some users will prefer to use Git in this manner, and it is quite common to see tutorials and resources online which refer to Git only by the command line interface.

On your PC or Mac, navigate to and open GitHub Desktop. You will need to sign into your GitHub account if you haven't already done this as part of "[GitHub Desktop Install Guide](#)". Using GitHub Desktop, we are going to copy the "my-git-tutorial" repository you created previously from the internet to our local computer. This operation is known as a **clone** in git. In GitHub Desktop, as shown in Fig. 9(a), go to File menu on the top and select "Clone repository..." option. As shown in Fig. 9(b), a new window will open asking you to select the

GitHub repository you like to clone. Select “my-git-tutorial” repository. Under “Local path” you may select a suitable place to save your cloned repository. We would recommend creating a folder name “GitHub” under Documents to save the “my-git-tutorial” repository as well as other repositories you will be cloning in the future. As shown in Fig. 10(a), GitHub Desktop will now perform a clone. Cloning can also be done directly through the GitHub web interface. As shown in Fig. 10(b), in your repository homepage, clicking the green “Clone” button will give you the option to directly open GitHub Desktop and clone this repository.

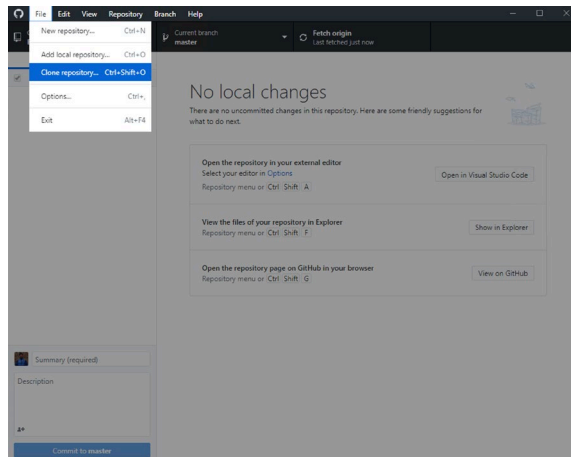


Fig. 9(a): Opening the cloning interface

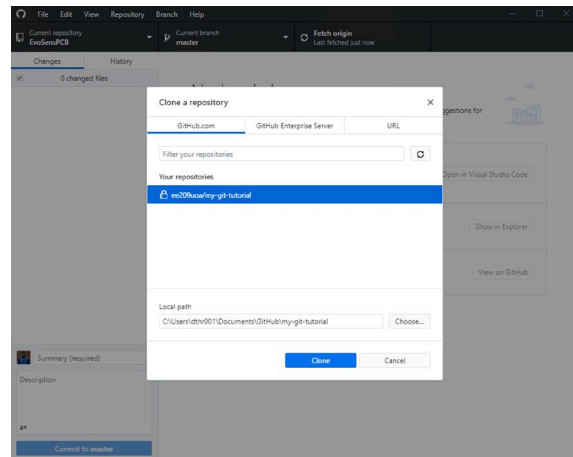


Fig. 9(b): Selecting repository to clone

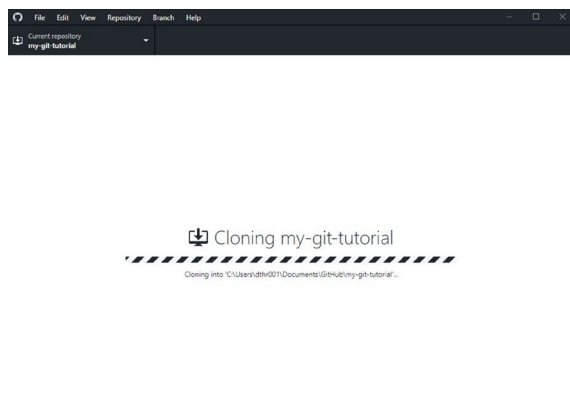


Fig. 10(a): Cloning process

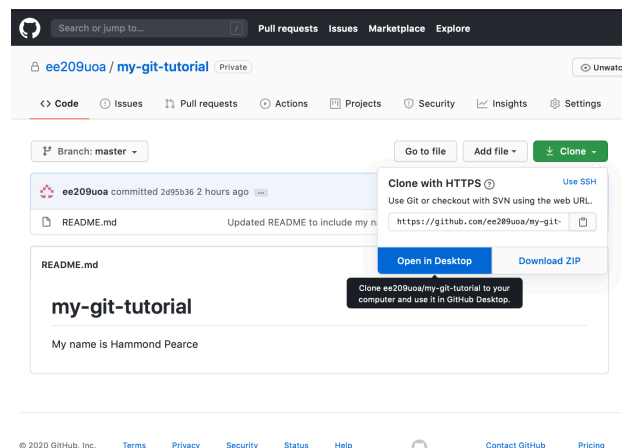


Fig. 10(b): Cloning from web

The GitHub Desktop GUI, as shown in Fig. 11, will now present you with key information about the repository saved on your computer (also called the local repository). This looks a little complicated but is actually quite straightforward. Let’s talk through it.

- The top bar labelled “1”, from left to right allow us to, select which repository we’re on, select which **branch** we’re on and **fetch & pull** updates from the **origin** or **push commit(s)** to the **origin**.
- In the left pane labelled “2”, we can see a summary of changes we’ve made to the local repository since our last **fetch & pull** or **clone**. It also let us view a **history** of changes made to files in the repository.
- The bottom section of the left pane allows us to add a **commit** message and **commit** changes to the local repository.

- The right pane labelled “3”, will show shortcuts if there are no changes made to the files in the local repository since our last fetch & pull or clone. If there are changes, as you will see later, the right pane will show details of these changes.

These are some new jargon words here, so let’s explain.

- In addition to allowing you to save different versions of files in a linear fashion, Git also allows for you to save different versions of the versions of files. In a sense, you can think of a repository’s history like a tree. Each **branch** of the tree represents which version of the file history you’re looking at. We won’t use this feature in this course, but it’s worth remembering.
- The default branch is called **master** and during this course we expect you to work on the master and not create any branches.
- When we download recent changes from GitHub, this is called a **fetch** (a **clone** downloads the entire repository, not just recent changes). A fetch is followed by a **pull** that merges the downloaded copy and the local copy.
- **Origin** refers to the version of the files on GitHub’s server.
- A **commit** is a snapshot of all the files in the repository at a specific time.
- Sending our **commit(s)** GitHub’s server is called a **push**.

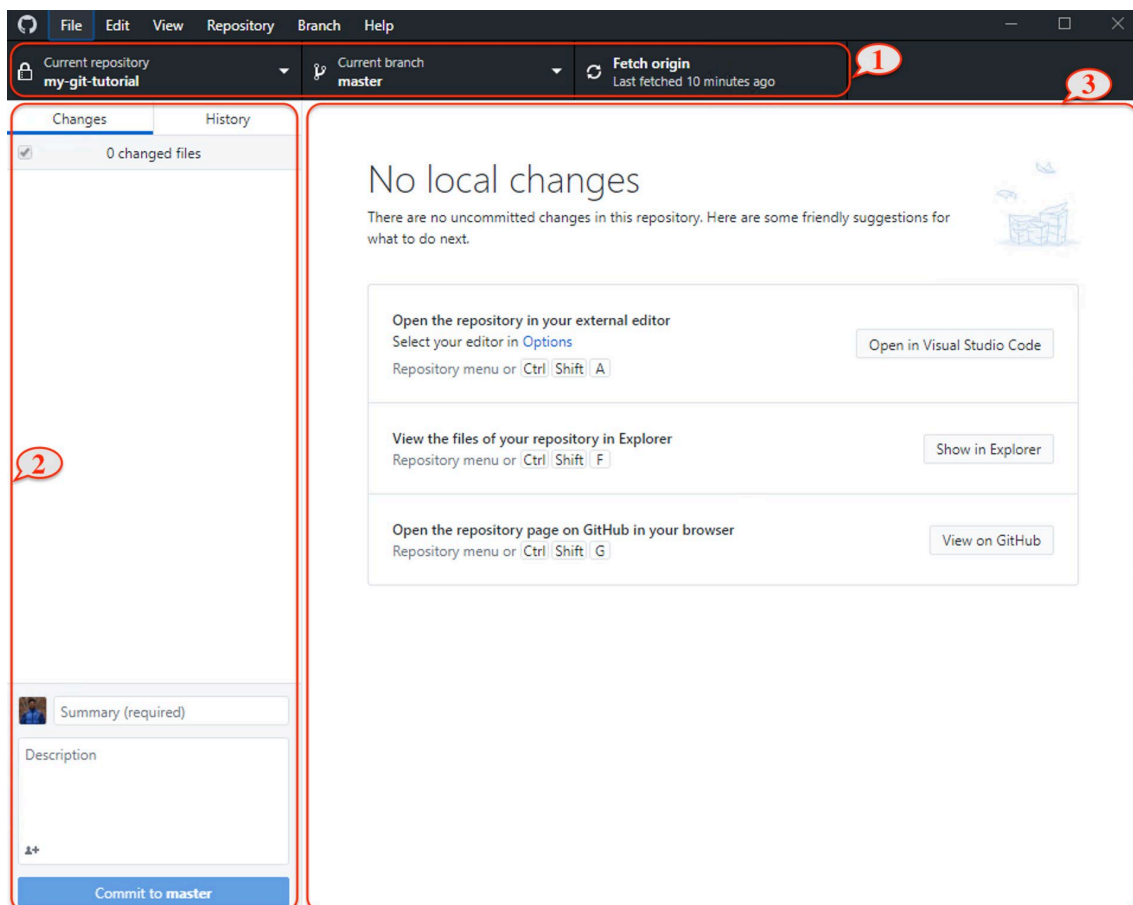


Fig. 11: The GitHub Desktop GUI

Editing Locally & Pushing

Let's now edit the README.md file we have in our local computer. As shown in Fig. 12(a), click on “Show in Explorer” button to reveal your repository in Windows Explorer, and then open the README.md file in a text editor. As shown by Fig. 12(b), we are using Notepad. Alternatively, you can use a modern editor like Visual Studio Code to edit the README.md file that is in our local repository we just cloned. Instructions on how to install Visual Studio Code can be found on Canvas in the [“Git Bash Install Guide”](#). As in Fig. 12(b), add the new line “This file has been edited on my computer.” to the README.md file and save the changes.

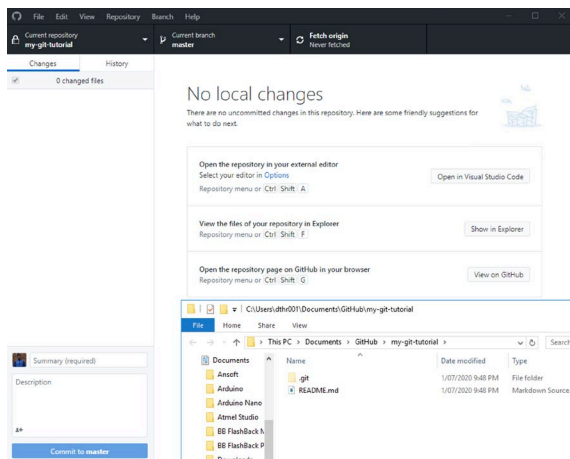


Fig. 12(a): Opening local folder

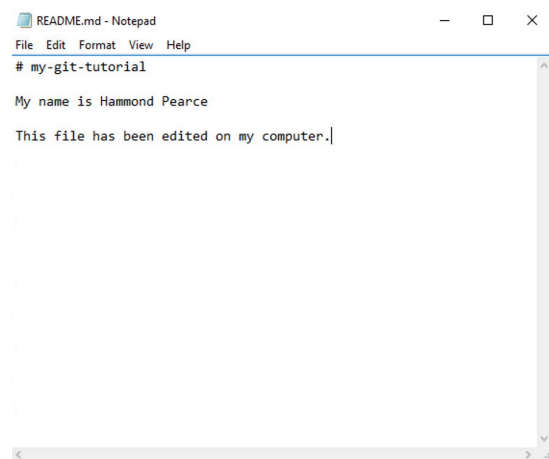


Fig. 12(b): Editing the local file

Return back to GitHub Desktop. You can observe that this now shows the README.md file has been updated. As shown in Fig. 13(a), the changes you have made are also shown. Add the commit message “Added some more text” and then press “Commit to Master”. This will create a snapshot of your repository and saves this version of your files locally in the local repository. If you were wondering where the information about your local repository is saved on your local computer, there is a hidden folder named “.git” in the same folder as the README.md file.

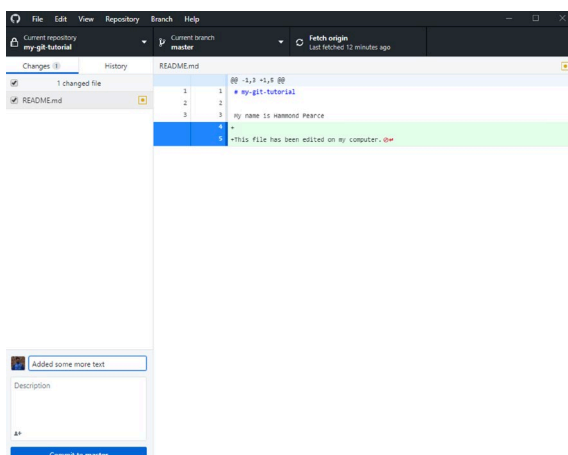


Fig. 13(a): Committing changes

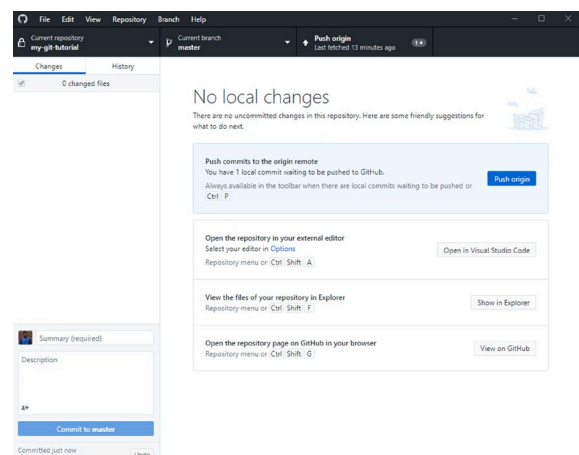


Fig. 13(b): Pushing to the origin

If you return to look at your repository on the GitHub server, you'll see that it hasn't changed yet. This is because committing created a local snapshot, but it **does not** automatically send this snapshot to the remote GitHub server. Instead, we need to send this ourselves. As we learnt earlier, sending our commit(s) to the remote server is called a **push**. Press either of the two "Push origin" buttons (blue button or the one in the top menu) that can be seen in Fig. 13(b). GitHub Desktop will now send your commit to the GitHub server. Once it has done so, wait a few seconds, and then on the browser refresh the homepage of your GitHub repository. You will now see the updated README.md along with the information about the commit you made.

Fetch and Pull

When you are working in a team project, multiple team members can edit a shared repository. Therefore, before adding your own changes, you need to make sure you update your own local repository with the latest on GitHub server. Let's emulate a scenario where a 2nd user has updated the remote repository on GitHub server. To do this, using the GitHub web interface (i.e. as in Fig. 5), edit the README.md file to add the line "Team members can use GitHub to collaborate." followed by a blank line to the README.md file as in Fig. 14(a). **Remember to add the blank line as it is needed for the next exercise (i.e. the README.md file should have 8 lines where last line is blank).** Add the commit message "A 2nd user added 2 new lines" and click the "Commit changes" button to commit.

Now go back to GitHub Desktop and click on "Fetch origin". Git will now check if there is a newer version of the repository. If there is, **fetch** will move the newer repository from the remote GitHub server to your computer. GitHub Desktop will now ask if you would like to "Pull origin" as shown in Fig. 14(b). A **pull merges** changes from remote repository with the local one, bringing in the changes done by the 2nd user and updating your local repository. As shown in Fig. 14(b), press "Pull origin". Finally, open the README.md file stored on your local computer using for example Notepad and confirm that it has been updated with the two new lines that were added by our hypothetical 2nd user. Though in this case the 2nd user edited the repository using the GitHub web interface, under normal use, it could be a team member had done some work prior to you on a different computer and pushed the work to the server.

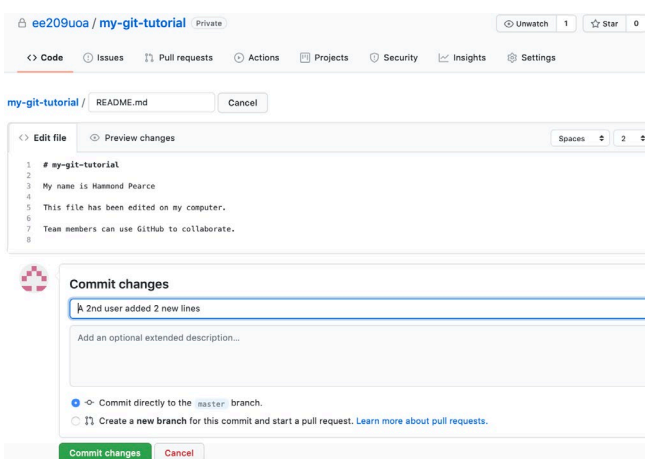


Fig. 14(a): 2nd user adding text

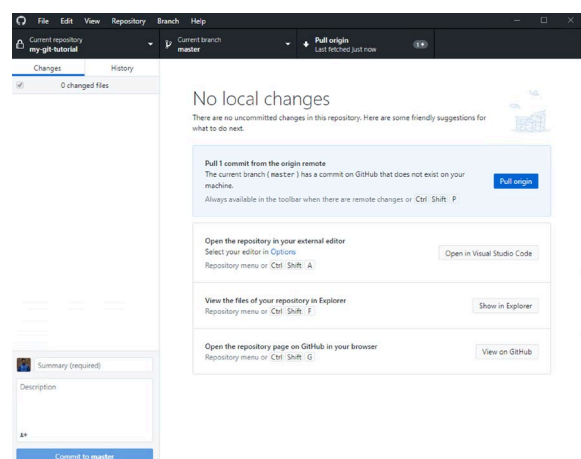


Fig. 14(b): Fetching updates

Merging

Let's now discuss one of the most powerful features of Git and GitHub. While saving snapshots of our files is extremely important, an even more important feature is the ability for multiple users to work on the same files at the same time. Compare this to services such as Dropbox, which restrict files to be edited by one user at a time. The way that Git manages this is through **branching**.

Let's first look at how a file might be changed over time. As an example, let's consider a shopping list. As shown in Fig. 15, every time we add/remove items to/from the shopping list, we save the changes to the list, commit and push. As we make these changes, the file changes and we keep storing snapshots of this file at each commit.

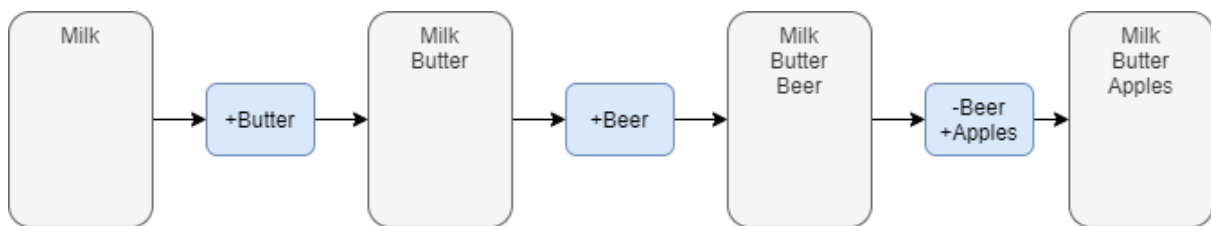


Fig. 15: A single user changing a file over time

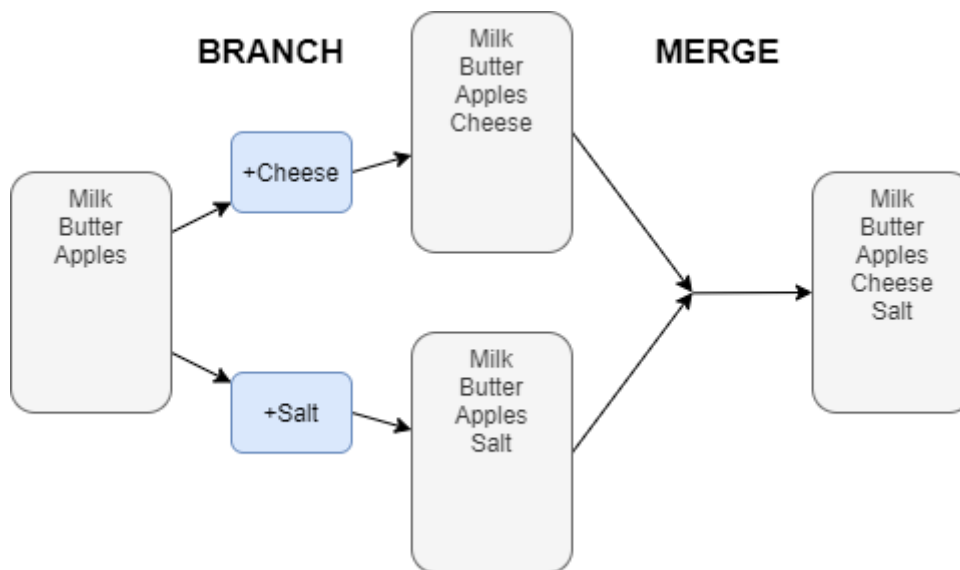


Fig. 16: Two users changing a file at the same time

Now, what happens if two users edit this file at the same time? Well, in Git, it manages this by allowing you to **branch**, creating two histories, and then providing you the tools to **merge** them together. This is shown in Fig. 16, where two users are simultaneously editing two copies of the original repository. One user is adding "Salt" to the shopping list while the other is adding "Cheese". These two copies can be **merged** at a later point in time as shown. Let's now emulate two users editing the same file by editing our README.md file in two different locations – simultaneously.

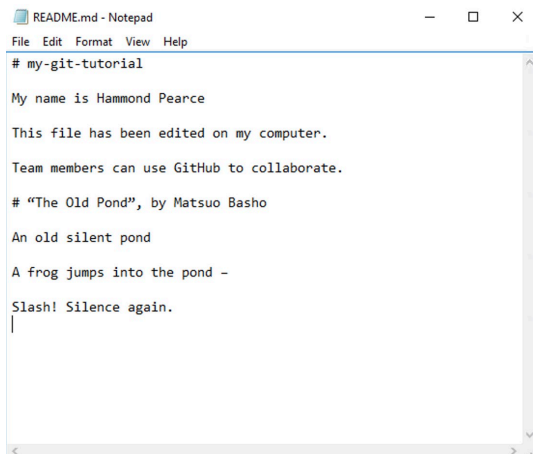


Fig. 17(a): Adding haiku

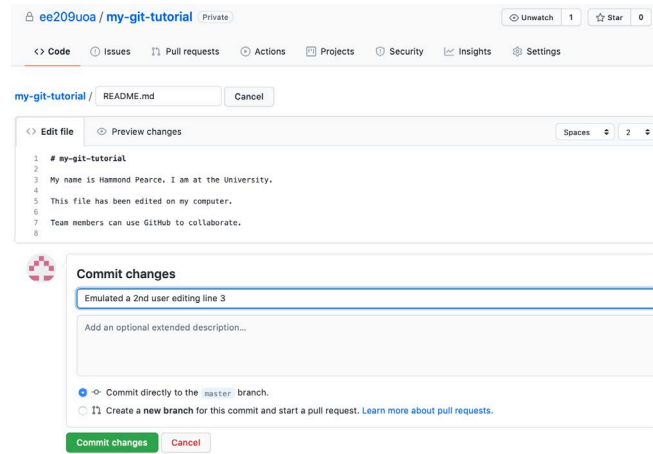


Fig. 17(b): Simultaneously editing online

First, let's create an additional change on the local version of the README.md stored on our computer. Add the famous haiku "The Old Pond" by Matsuo Basho given below to the README.md stored on your computer.

```

# "The Old Pond", by Matsuo Basho

An old silent pond

A frog jumps into the pond -

Slash! Silence again.

```

As shown in Fig. 17(a), we are using Notepad to edit the file, but you are welcome to use an editor you are familiar with. Save the changes you just made to the local README.md file. Then go to GitHub Desktop add the commit message "Added haiku from local computer" and then press "Commit to Master". **DO NOT push the changes to GitHub server yet** since we are trying to emulate a scenario where two users are editing the same repository simultaneously. If it is easier, assume the 1st user edited the local repository to add the haiku.

Now let's emulate a 2nd user editing a copy of the same original repository you started with - in our case the original repository had a single README.md file with 8 lines as shown in Fig. 14(a). Assume the 2nd user edited this README.md file online using the GitHub browser interface. This is so that we can emulate the 2nd user updating the origin (i.e. original copy of the repository on GitHub server) before the 1st user does. Now that you know what needs to be done, using the GitHub web interface (i.e. as in Fig. 5), edit the README.md file to change line 3 from "My name is Hammond Pearce" to "My name is Hammond Pearce. I am at the University.". This is shown in Fig. 17(b). **Be very careful to only change this line.** Add the commit message "Emulated a 2nd user editing line 3" and click the "Commit changes" button to commit.

Return back to GitHub Desktop and click "Push Origin" to send the changes done by the 1st user to the GitHub server. As shown in Fig. 18(a), you will be presented with a message. The key information is in the message heading, there are "Newer commits on remote". This means that on the remote GitHub server there is work that is newer/different than what you have on your machine. This is because of the change we committed using the GitHub web interface emulating a 2nd user editing the same repository. A similar scenario could happen if your friend had done some work simultaneously to you on a different computer and pushed their work to the server before you. Press "Fetch origin". Recall that **fetch** moves work from the remote GitHub server to your computer. However, this action does not combine the two copies of the work (local copy and the latest copy moved from the remote server) together. Therefore, you will be presented with the message in Fig. 18(b), giving the option to do a **pull**. A **pull merges** remote commits with local commits, combining the changes on

our local computer and creating a single up to date copy of the work on your local computer. As shown in Fig. 18(b), press “Pull origin”.

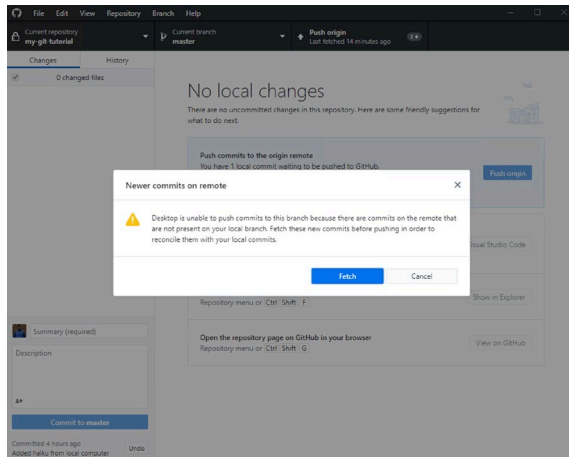


Fig. 18(a): Request to fetch

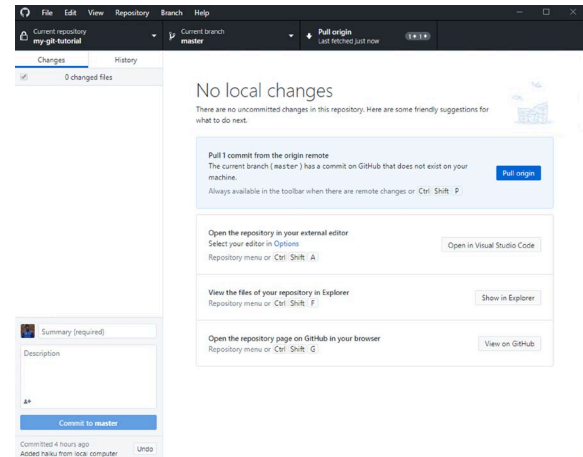


Fig. 18(b): Merging changes

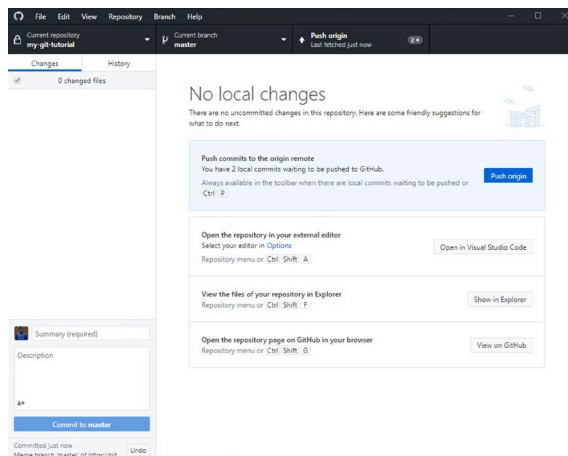


Fig. 19(a): Pushing changes

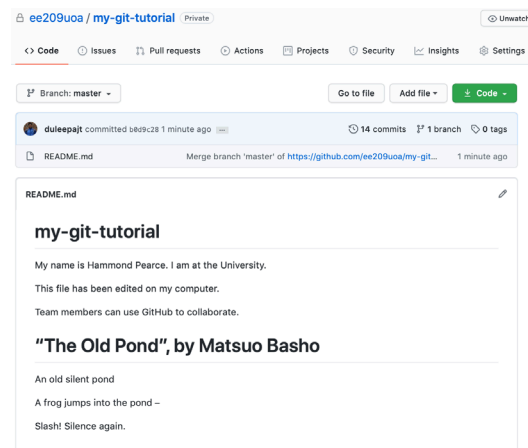


Fig. 19(b): Pushing to the origin

As our changes were in two different locations in the file, Git is able to merge these two changes automatically, and we are now presented with the option to push changes to the origin as shown by Fig.19(a). Observe that it says that there are now 2 commits to send. One of them is the work from earlier adding the haiku. The other commit is the pull you just did which merged our work with the online work. Press “Push origin”, wait till it completes. Wait a few more seconds, and then on the browser refresh the homepage of your GitHub repository. You will now see the README.md updated with all changes made (i.e. 1st user’s and 2nd user’s edits are integrated) as in Fig. 19(b).

Handling Merging Conflicts

Sometimes it is not possible for Git to automatically merge files together. This usually happens when two members of a group have worked on the same file in the same location. When this occurs, you will have to manually to solve the conflicts and instruct it which changes you wish to keep. Let’s now emulate a merge conflict, so that we can learn how to resolve a conflict. To do this, similar to the previous exercise, we are

again going to emulate two users editing our “my-git-tutorial” repository. Both users in this case will be editing the same line.

As before, first edit the README.md file stored on your local computer using Notepad or another suitable editor (emulating again the 1st user making a change to the local repository). We are going to change the formatting of the 1st line in our README.md file as shown in Fig. 20(a). Change “# my-git-tutorial” to “# My-Git-Tutorial” by capitalising first letter of each word. Save the changes you just made to the local README.md file. Then go to GitHub Desktop add the commit message “Modified title from local computer” and then press “Commit to Master”. **DO NOT push the changes to GitHub server yet** since we are trying to emulate a scenario where a 2nd user simultaneously editing the same line of the file.

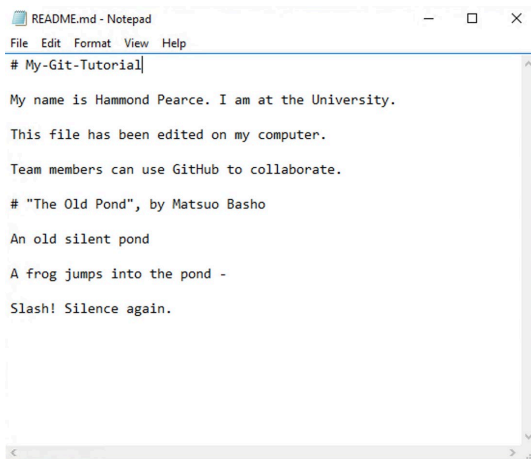


Fig. 20(a): Changing 1st line formatting

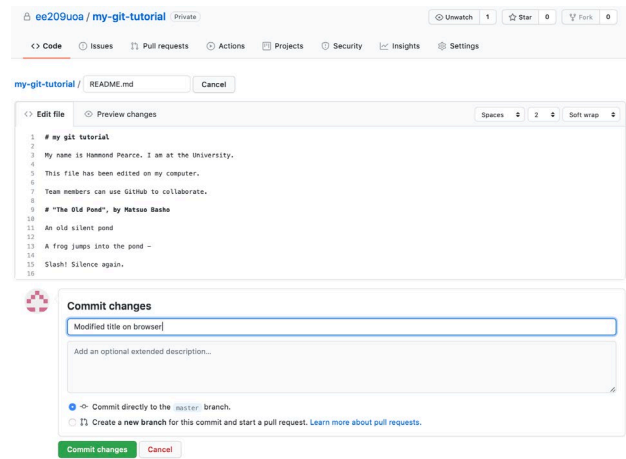


Fig. 20(b): Pushing to the origin

As before, to emulate a 2nd user editing a copy of the same original repository, using the GitHub web interface, edit the README.md. You’re going to change the 1st line from “# my-git-tutorial” to “# my git tutorial” by removing the dashes between words. As shown in Fig. 20(b), add the commit message “Modified title on browser” and click the “Commit changes” button to commit.

Return back to GitHub Desktop and click “Push Origin” to send the changes done to the local copy of the repository by the 1st user to the GitHub server. As before, it will again warn you there are newer commits on the remote (i.e. same message as in Fig. 18(a) will be displayed). Press ‘Fetch’ to continue. As in Fig. 18(b), it will now present the “Pull origin” button. Press the “Pull origin” button to continue. Since the same line has been edited in both copies of the repository, Git cannot automatically decide which version to use. Therefore, this time, instead of Git solving the merge automatically, we are presented with the message shown by Fig. 21(a).

As the message in Fig. 21(a) states, we need to open the README.md file and edit it manually to resolve this **merge conflict**. From the drop-down menu select which text editor you are going to use to open the README.md file. As shown in Fig. 21(b), we are using Notepad to open the README.md file and resolve the conflict. You’ll notice that in the new README.md file it has both versions of the edits (i.e. “# My-Git-Tutorial” and “# my git tutorial”). These two lines are separated by magic characters <<<<<<<, =====, and >>>>>>> followed by a long hexadecimal string. This hexadecimal string is what is known as a “commit hash”, which is what Git uses internally to identify the different commits. There is only one conflict here, so only one section for us to solve. For files that had more conflicts you would need to go through and solve each conflict individually. For us here, we probably want both changes, so let’s edit the 1st line to “# My Git Tutorial” as shown in Fig. 22(a). Also, delete all the magic characters to indicate you have resolved this merge conflict. Finally save the README.md file.

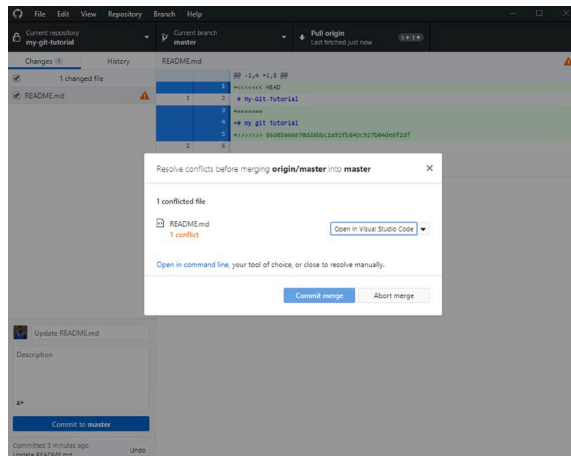


Fig. 21(a): Warning about conflict

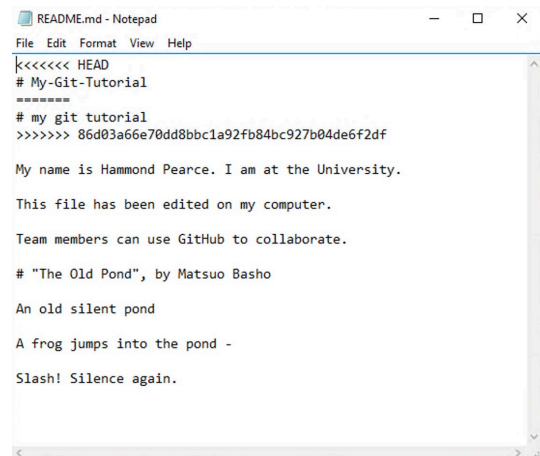


Fig. 21(b): File showing conflict

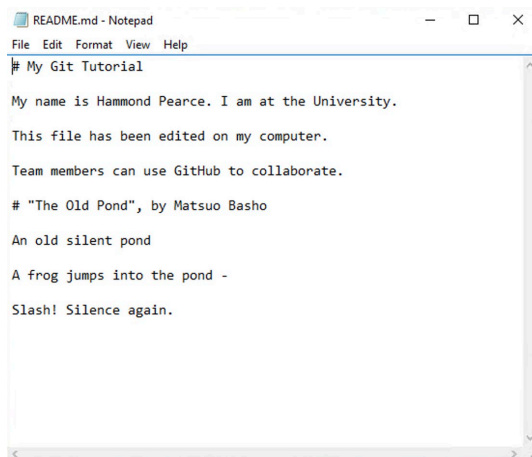


Fig. 22(a): Resolving conflict

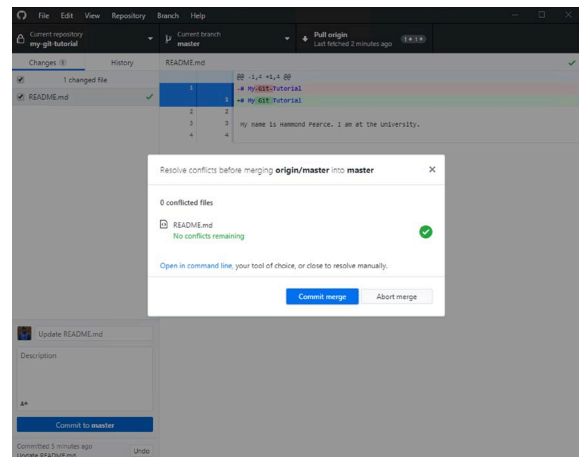


Fig. 22(b): Committing merge

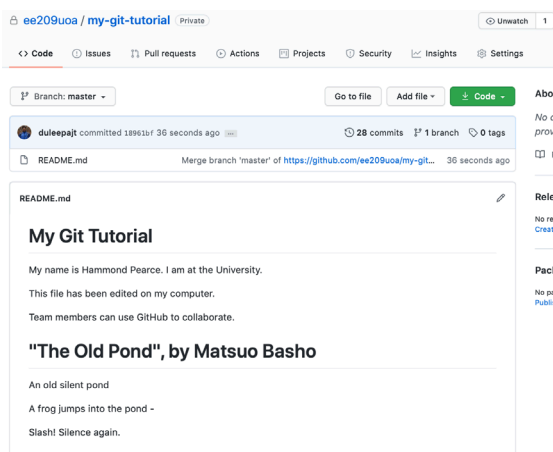


Fig. 23(a): Updated repository

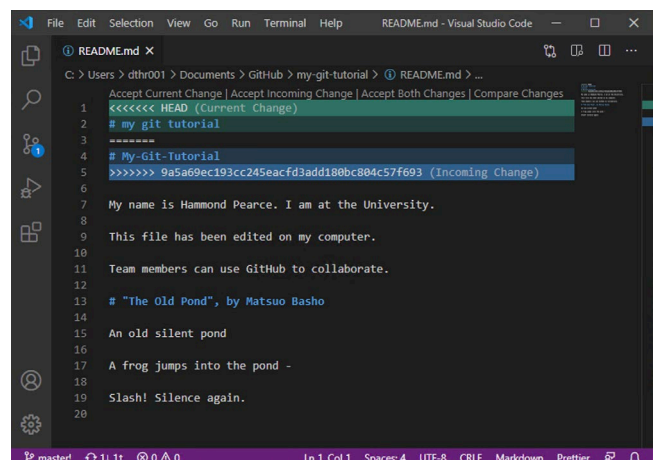


Fig. 23(b): Using VSC

Return back to GitHub Desktop. As shown by Fig. 22(b), you will notice that the message has changed stating there are no more conflicts to resolve. Press “Commit merge”, and then press “Push origin” to update the remote repository. Wait a few more seconds, and then on the browser refresh the homepage of your GitHub repository. You will now see the README.md updated with all changes made as in Fig. 23(a).

As shown in Fig. 23(b), if you had used a more modern editor, like Visual Studio Code in this example, it will provide you a few shortcuts to help you resolve merge conflicts. For example, if we only wanted to keep the change we made to the local file (i.e. ignore the change made through the web interface), then we could have clicked on “Accept Current Change”. This would remove all the magic characters and leave the 1st line as “# my git tutorial”. Alternatively, as you did with Notepad, you can also manual edit the file in Visual Studio Code.



THE UNIVERSITY OF
AUCKLAND
Te Whare Wānanga o Tāmaki Makaurau
NEW ZEALAND

Dep. of Electrical, Computer & Software Engineering

The University of Auckland

Building 405, Room 631

20, Symonds Street

Auckland, New Zealand

T +64 9 923 9634

W auckland.ac.nz