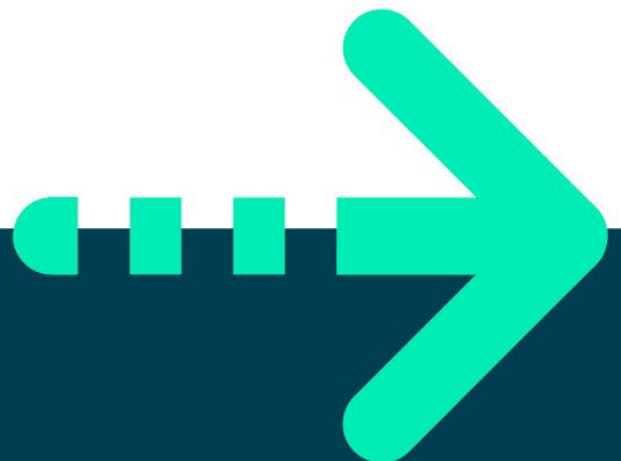




# **SDL3 Module 5:**

## **Git**

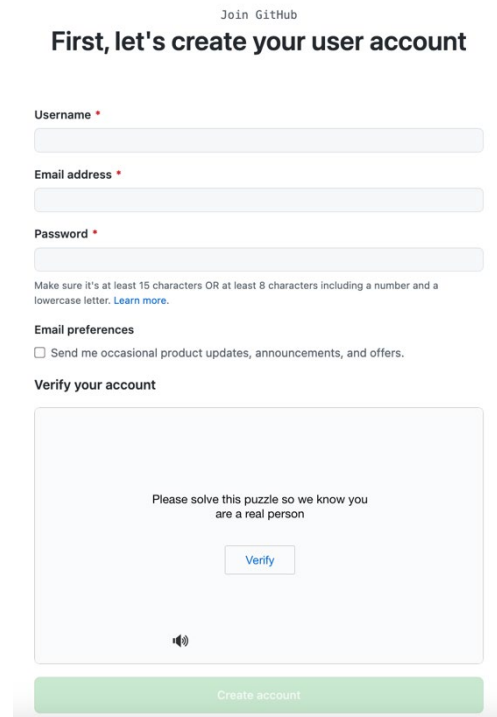
### **Exercise Guide**



# Exercise 1 – Getting Started

## Part 1 – Creating a Git Account

1. Open your browser and navigate to <https://github.com/join>



Join GitHub

**First, let's create your user account**

Username \*

Email address \*

Password \*

Make sure it's at least 15 characters OR at least 8 characters including a number and a lowercase letter. [Learn more.](#)

Email preferences

☐ Send me occasional product updates, announcements, and offers.

Verify your account

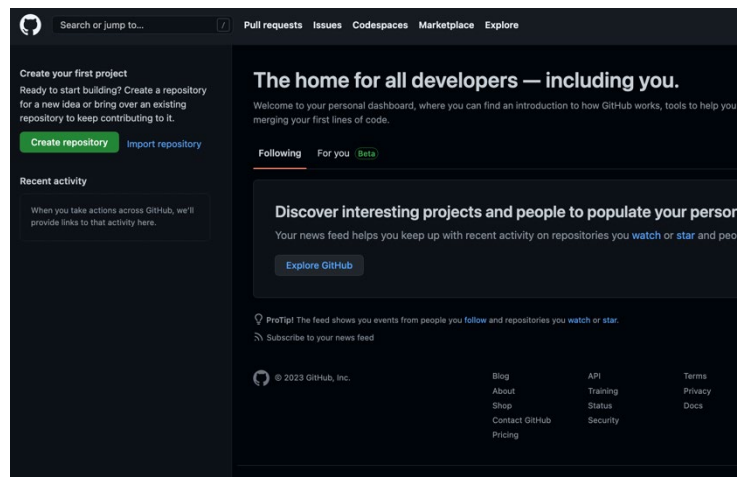
Please solve this puzzle so we know you are a real person

Verify

Create account

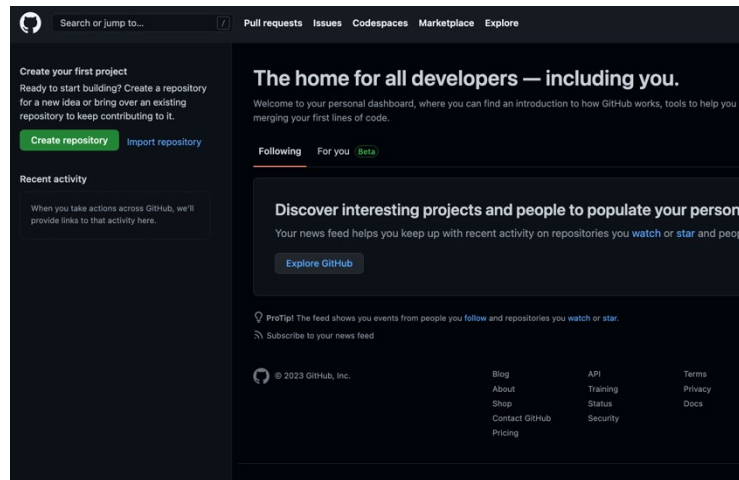
2. Fill in the form presented with a username, email address and password.
3. Once signed up, you may be asked to enter a verification code that you will receive via your email.
4. Once entered, you can skip the personalisation options or fill them in – it's up to you!

Once completed, you should see a screen similar to the below:



## Part 2 – Creating a Git Repository

1. Now that we have an account with Git, we need to create a repository! Select the green button on the left-hand side of the GitHub homepage.



2. Fill in the details on the repository and use the following information:
  - a. Repository name: Feel free to call this what you wish, but you can default to Example-Repo. Try not to make the name too long!
  - b. Leave the description field blank.
  - c. Ensure the repository is public.
  - d. Tick the box to add a README.md file to the repository.
  - e. Leave the .gitignore and license options default.
  - f. Select the Create Repository button.

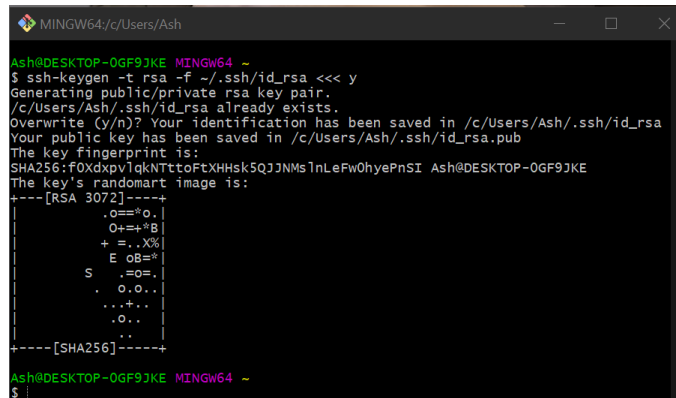
Congratulations, you've created a repository on GitHub! Let's clone this so we can begin to use it.

## Part 3 – Cloning the repository

We have an account, and now we have a repository, we can clone it down to our local machine. First, let's set up the environment!

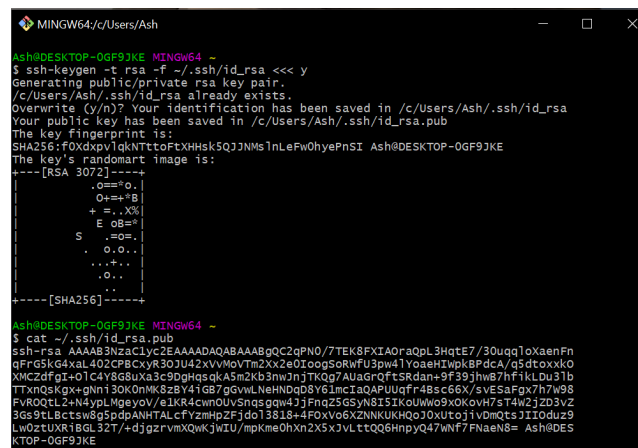
1. Open Git Bash on your machine and change directory to your User Home with the command:  
`cd [YOUR USER HOME]/`
2. In this folder, we'll create a new directory called git-project with the command:  
`mkdir git-project`
3. Change directory into this folder with the command:  
`cd git-project`
4. Run the `dir` command – what is produced?

- Before we can clone down the repo, we need to add an SSH key. In Git Bash, run the below command:  
`ssh-keygen -t rsa -f ~/.ssh/id_rsa`
- Once you hit enter, you may be prompted for a passphrase. Simply hit enter and continue without a passphrase.



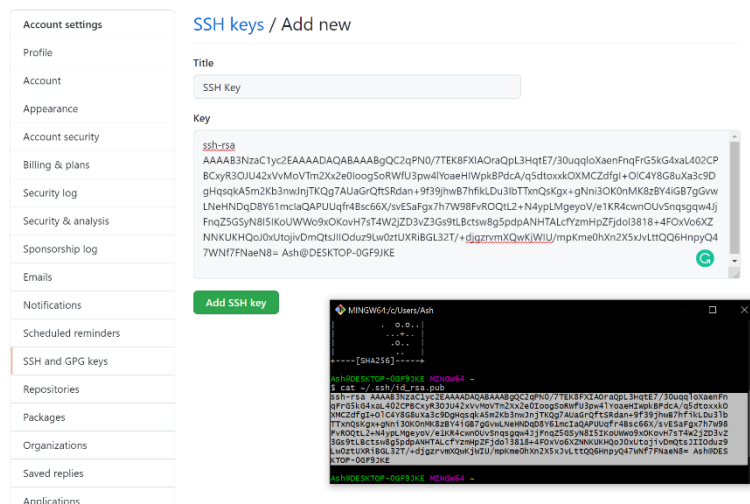
```
MINGW64/c/Users/Ash
Ash@DESKTOP-0GF9JKE MINGW64 ~
$ ssh-keygen -t rsa -f ~/.ssh/id_rsa <<< y
Generating public/private rsa key pair.
/c/Users/Ash/.ssh/id_rsa already exists.
Overwrite (y/n)? Your identification has been saved in /c/Users/Ash/.ssh/id_rsa
Your public key has been saved in /c/Users/Ash/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:F0XdxpVlqkNTtToFTXHHsk5QJ3JNMs1nLeFw0hyePnSI Ash@DESKTOP-0GF9JKE
The key's randomart image is:
+----[RSA 3072]-----+
|.o==.o.|
|O+==+B|
|+ =.X%|
|E oB==|
|S .o=|
|.O.O..|
|...+..|
|...+..|
|...+..|
|...+..|
+----[SHA256]-----+
Ash@DESKTOP-0GF9JKE MINGW64 ~
$
```

- Next, run the command below and copy the output:  
`cat ~/.ssh/id_rsa.pub`

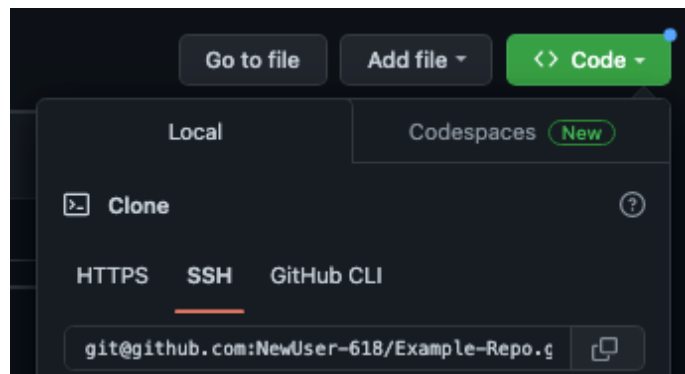


```
MINGW64/c/Users/Ash
Ash@DESKTOP-0GF9JKE MINGW64 ~
$ ssh-keygen -t rsa -f ~/.ssh/id_rsa <<< y
Generating public/private rsa key pair.
/c/Users/Ash/.ssh/id_rsa already exists.
Overwrite (y/n)? Your identification has been saved in /c/Users/Ash/.ssh/id_rsa
Your public key has been saved in /c/Users/Ash/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:F0XdxpVlqkNTtToFTXHHsk5QJ3JNMs1nLeFw0hyePnSI Ash@DESKTOP-0GF9JKE
The key's randomart image is:
+----[RSA 3072]-----+
|.o==.o.|
|O+==+B|
|+ =.X%|
|E oB==|
|S .o=|
|.O.O..|
|...+..|
|...+..|
|...+..|
|...+..|
+----[SHA256]-----+
Ash@DESKTOP-0GF9JKE MINGW64 ~
$ cat ~/.ssh/id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQGC2qPNO/7TEK8FXIAOraQpL3HqtE7/30uqql0xaelFn
qFRG5K64xal402CPBCxyR30JU42xVvMvTm2Xx2e0toog5oRwFU3pw41YoaehIwPk8PdcA/q5dtoxxk0
VNC2dfgt+01c4y8G8uXa3c9dgloqgkASm2Kb3mWjTKQg7AUaGrQttSRdan+9F39jhw7HfKLDu31b
TTXNqKgX+gNn130K0Nk8zBY41G87g0vWLnEHNDQ8Y61mcIAQAPUuqf48sc66X/svESaFgx7h7W98
FVRQTL2+N4yplMgeyov/e1KR4cwnOUvSnqsgqW4jFngZ5GSyN8I5IKoUw09x0KovH7sT4w2jZD3v2
3G9tLBctsw8g5pdpANHTALcfYzmHpZFjdo13818+4FOxVo6ZNNKUKHQJ0xUtojiVDMQtsJIIoduz9
LW0ztUxR18GL32T/+dJgzrvmXQwKjWU/mpkme0Hxn2X5xJvLttQ6HnpYq47wNf7FNaeN8= Ash@DES
KTOP-0GF9JKE
```

- Navigate to this link: <https://github.com/settings/ssh/new> - paste the output of the command ran in step 7 like so, and select **Add SSH key**:



9. Now, we can finally clone down the repository we made! Go back to the repository you created in GitHub, and in the green drop-down for code and choose SSH. Copy the URL presented:



10. Go back to Git Bash and clone the repo down with the command:  
`git clone [URL you copied from the repo]`
11. We can now run the **ls** command and see that we have a folder in here – change directory into it with the **cd** command and run another **ls**. You will see the README.md file we had on our repository, meaning you've cloned your repository!

## Exercise 2 – Adding files

### Part 1 – Adding to the remote repository

1. Open the folder for the cloned repository from exercise 1 in your File Explorer.
2. Create three files in this folder – File1.txt, File2.txt, File3.py
3. Enter some basic text into the two text files, and a line or two of basic Python code (such as a print line or some variable declarations) into the Python file.
4. In Git Bash, navigate to the folder in which your cloned repository is, and run the command `git status`. What do you see?
5. Edit your README.md with some text that explains what this repository is.
6. Add all the files to staging with the command:  
**`git add .`**
7. Run a commit command explaining with a message what this commit is, such as the below:  
**`git commit -m "first push to the repository"`**
8. Push the changes to your repository with the command:  
`git push origin main`

### Part 2 – Adding more files

1. Create a new file called file.java and enter some basic Java into the file.
2. Repeat the steps to add, commit, and push these changes to your repository.

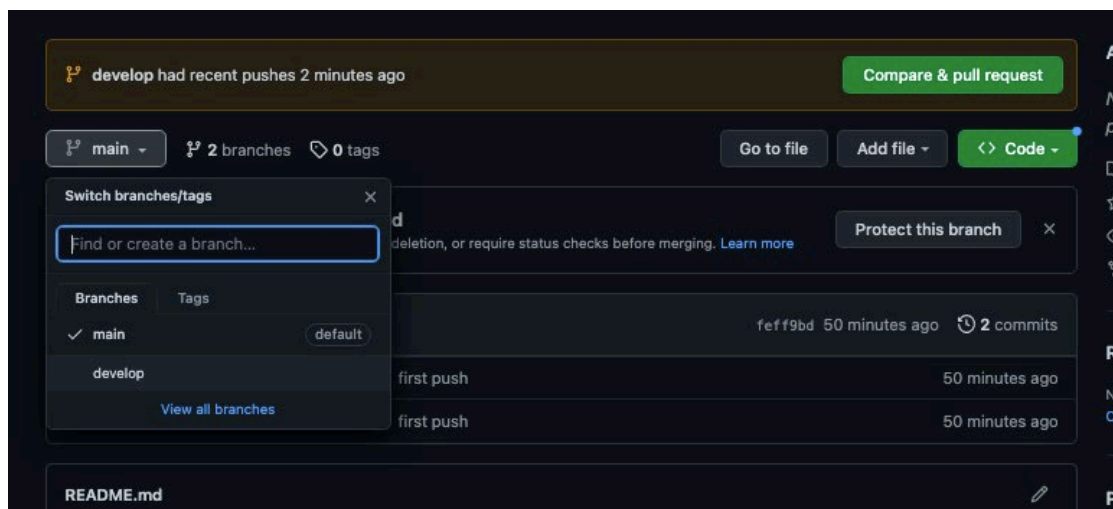
## Exercise 3 – Branching

### Part 1 – Creating the branch

1. In Git bash, run the below command. What do you see?  
`git branch`
2. Let's create a new branch called develop by running the below command:  
`git branch develop`
3. Run the command to see all branches in your repository again. You'll notice you are still working on the main branch. Let's switch, or checkout, the develop branch with the below command:  
`git checkout develop`

### Part 2 – Working on a new branch

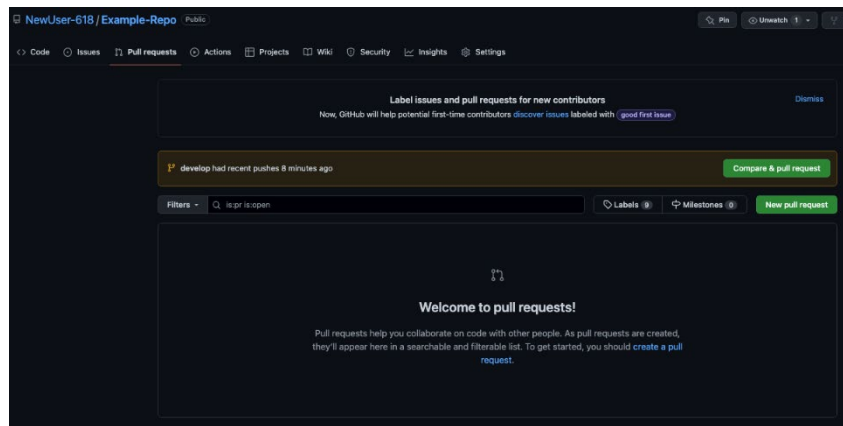
1. On the develop branch, create a new file. Feel free to edit this file.
2. Once you have created and edited a file, add this file to staging and commit it to repository with a message specifying that this is a change on the development branch.
3. We will now push this change to our remote repository – run the below:  
`git push origin develop`
4. Go to your GitHub repository. You may notice an info message above your files. Select the branch dropdown box and switch to the develop branch.



What are the differences you see between your main and develop branches? Why is this the case?

## Exercise 4 – Pull request

1. On GitHub, go to your repository and navigate to Pull Request via the main bar:



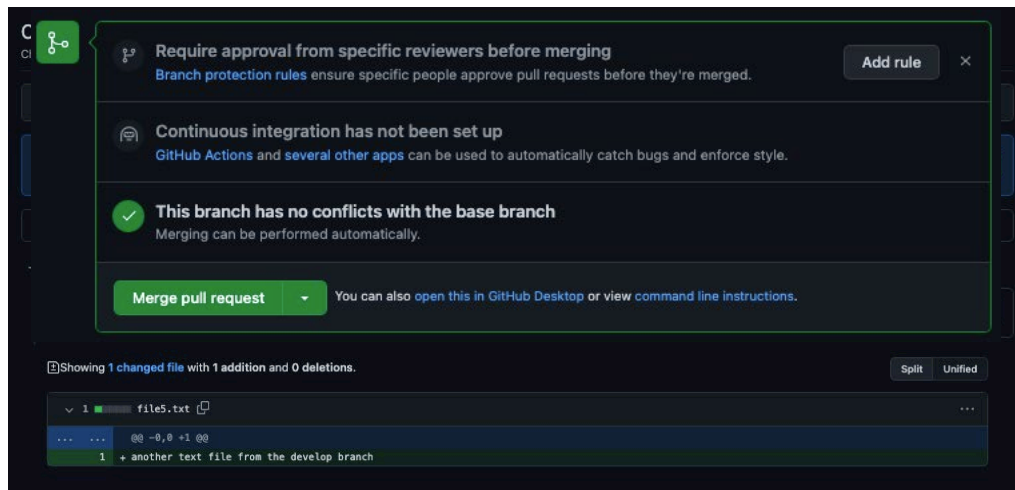
2. Select the 'New Pull Request' button on the right-hand side of the screen.
3. You will be presented with a screen asking to compare changes. Just below the main title for the page, you'll see two drop down boxes. Ensure that these boxes have the main branch chosen as the base, and the develop branch chosen as the head.

We are going to combine the content in the develop branch into the main branch. You should see a green message with a tick next to the boxes saying 'able to merge' if done correctly.





4. Once changed, select Create Pull Request.
5. On the following screen, you can leave a comment describing the change, and select the green button at the bottom of the page when done.
6. You'll then be brought to another screen that will allow you to confirm your merge. Select the green button, confirm the merge, and combine the branches.



7. Go back to your repository code base and have a look at the main branch. You should see the contents of the develop branch in your main branch!

## Exercise 5 – Merge conflicts

### Part 1 – Creating a new repo and file

1. We're going to create a merge conflict in a new repository. To do this, navigate to Git Bash and create a new folder in the terminal with the command:  
`mkdir git-merge-conflict-example`
2. We've created a folder called git-merge-conflict-example with the `mkdir` command. Let's go into that directory with the `cd` command:  
`cd git-merge-conflict-example`
3. Instead of cloning down a repository, we're going to initialise this folder as a Git repository. Run the below command:  
`git init`
4. Next, let's add a file called `hello.txt` into the folder through the terminal. You can either create the file via the file explorer in the relevant folder, or run the below command to create an empty file:  
`touch hello.txt`
5. Let's add some text into this – something like 'Hello World!' Again, you can do this via the file explorer and open the file, make your changes, and save. You can also do this in the terminal via the vim tool:  
`vim hello.txt`

Note: if you are using vim, hit the **i** key to **insert** content, then enter what you wish to enter. Hit the **ESC** key, followed by the combination of **:wq** to save the file and quit the vim tool.

6. Let's add this file to staging – run the command `git add hello.txt`
7. Let's then commit this with the below command:  
`git commit -m "first commit"`

### Part 2 – Making a new branch

1. When we encounter merge conflicts, it will be because there is a mismatch of tracking and updates between branches working on the same file.

To create a merge conflict, let's checkout and create a new branch with the below command:

```
git checkout -b new-branch
```

2. Using either the file explorer or the vim command and the keys mentioned in the previous part, open the `hello.txt` file and add a new line below what is already there that says 'making a change to the file'

3. Add this file and commit it with the below commands:  

```
git add hello.txt  
git commit -m "made a change to the file"
```
4. Run the below command to see the contents of the hello.txt file:  

```
cat hello.txt
```

### Part 3 – Making the merge conflict

Now that we've made a change to the file on our new-branch branch, we'll need to make a change to the file on our default main/master branch to cause a merge conflict.

1. First, run the `git branch` command to see which branches you have available. You should have either a main or master branch alongside the new-branch branch you are currently working on.
2. Switch to the other branch with either `git checkout master` or `git checkout main` depending on the name of the branch.
3. Open the file hello.txt and add a new line of text saying, 'making a bigger change'.
4. Run the below command to see the contents of the hello.txt file – you'll notice that it's different to the content seen on the new-branch branch:  

```
cat hello.txt
```
5. Let's add and commit the changes on this branch with the below commands:  

```
git add hello.txt  
git commit -m "modified the hello.txt file"
```
6. Run the below command:  

```
git merge new-branch
```

You'll notice that, after some operation, Git will present a message similar to the below:

```
CONFLICT (content): Merge conflict in hello.txt  
Automatic merge failed; fix conflicts and then commit the result.
```

### Part 4 – Resolving the conflict

1. Run the below command to see the contents of the hello.txt file after the attempted merge:  

```
cat hello.txt
```
2. To resolve this merge conflict, we need to decide which change to keep, and then delete the content inserted by Git for the merge conflict. Open the file again, either in your File Explorer or vim.

3. Remove the line 'making a change to the file' from the file
4. Remove the lines inserted by Git – these should be the line of less-than signs (<<<<<< HEAD), the line of equal signs (=====), and the line of greater-than signs (>>>>>> new-branch)
5. Save the file.
6. Run the below command to see the contents of the hello.txt file after resolving the merge conflict:  
`cat hello.txt`
7. Let's add and commit the changes on this branch with the below commands:  
`git add hello.txt`  
`git commit -m "resolved the merge conflict"`

Congratulations, you resolved a merge conflict in Git!

## Exercise 6 – Reverting in Git

1. Create a folder called revert-exercise with the below command:  
`mkdir revert-exercise`
2. Change directory (`cd`) into the revert-exercise directory.
3. Initialise the folder as a git repository with the below command:  
`git init`
4. Create a new file called test.txt with the below command:  
`touch test1.txt`
5. Stage the file and commit, with an explicit commit message:  
`git add test1.txt`  
`git commit -m "added test1.txt to the commit"`
6. Repeat the previous two steps (using a different file name each time (e.g., test2.txt, test3.txt) until you have done 3 commits. Make sure the commit messages are distinct!
7. Now check the log history for the branch you are on with the below command:  
`git log [branch name]`
8. Use the `git revert HEAD` command to 'undo' the last commit you did. You'll be brought to a vim screen - you can exit by entering the `:` and `q` keys (i.e. `:q`).
9. Use the `ls` command to see what happened – how many files are present?
10. Check the history for the branch you're working on again. What is the output of this command after using the revert command?

