

# GitHub Tutorial – Basics

Dr. Xianhui Che  
x.che@qmul.ac.uk

## Objectives:

- Understand the basic GitHub web interface
- Register GitHub account
- Learn to create repository, commit, fork, pull request, branching, etc
- Learn how GitHub handles merge in terms of file management
- Get useful tips on various types of commits

## Step 1. Register a GitHub Student Developer Pack

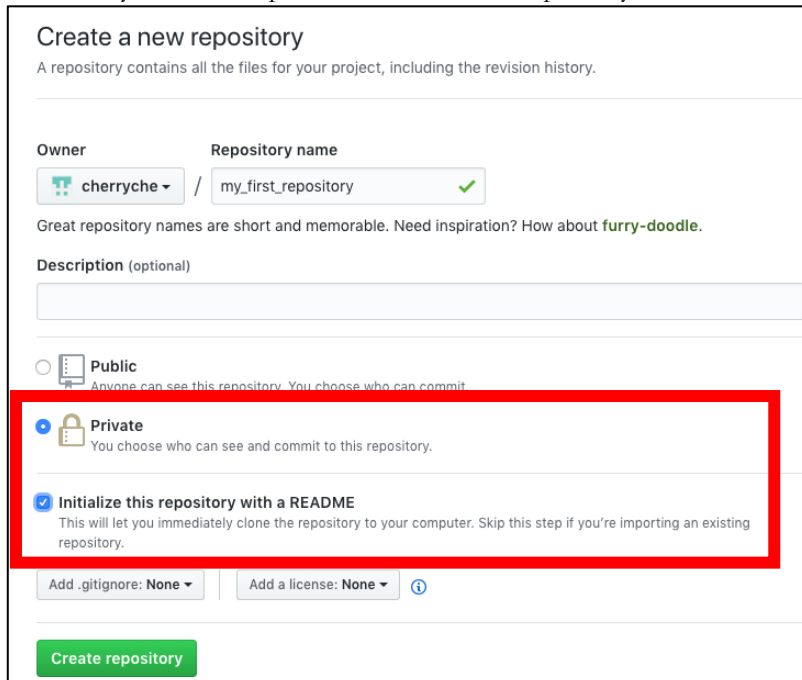
By default, the repositories you create in GitHub is public, which means everybody in the world can access them. It normally costs a subscription fee to create private repositories. The good news is it is free for students.

Visit <https://education.github.com/pack>, and register for a Student Developer Pack using your **xxxx@qmul.ac.uk** email address.

## Step 2. Create a Private Repository

Follow these instructions:

1. In the GitHub homepage, click “New Repository”.
2. Give a name to your repository and description (optional). Choose “Private” in the next section. And make sure you tick the option of “Initialize this repository with a README”.



**Create a new repository**  
A repository contains all the files for your project, including the revision history.

Owner: **cherryche** / Repository name: **my\_first\_repository** ✓

Great repository names are short and memorable. Need inspiration? How about **furry-doodle**.

Description (optional):

☐ Public  
Anyone can see this repository. You choose who can commit.

☒ **Private**  
You choose who can see and commit to this repository.

☒ **Initialize this repository with a README**  
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: **None** | Add a license: **None** ⓘ

**Create repository**

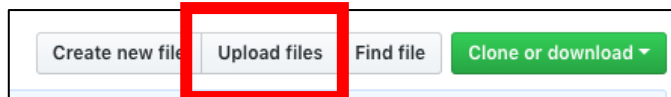
3. Your repository now has an official URL with something like:  
[https://github.com/cherryche/my\\_first\\_repository](https://github.com/cherryche/my_first_repository)

### Step 3. Commit to the Private Repository

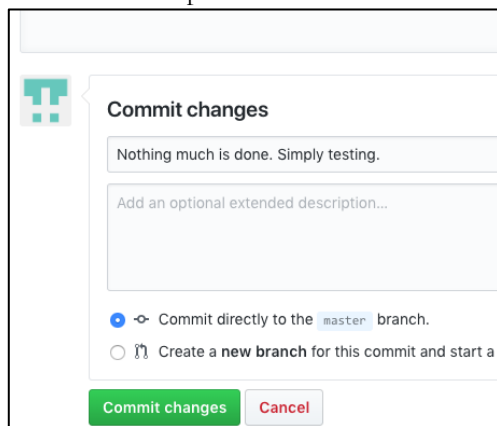
For the learning purpose, you may use any software files that you wish to upload to the GitHub repository. (It can even be a text file.)

Follow these steps:

1. Enter your repository.
2. Click “Upload Files”.



3. Either use “Choose your files” option or simply drag the files to the box. Dragging allows you to upload a whole folder of files at once.
4. Add some descriptions. And click “commit changes”.



5. In the future, should you wish to commit again, simply follow the same procedure to upload files.

This practice should go in parallel with the Agile management during your app development process.

The milestone stages to commit (i.e. upload) a repository are the moments when you complete a newer version of the software project (i.e. at the end of each sprint). However, it is advised that you upload your work whenever you can for your own sake – you will personally benefit from it for these two reasons:

- Just in case something goes wrong with your memory stick or machine, you always have a backup in the cloud.
- Suppose you had a good copy of your software project files that worked perfectly fine, then you made some changes and it stopped working and it would not revert back to the previous working state even if you cancelled all the changes (no matter how hard you tried). If you are a good version control practitioner, you can always count on GitHub to download the previous version (or any former version).

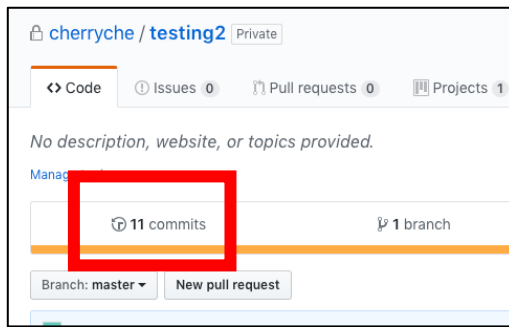
Therefore, every time after doing some hard work, please upload them to GitHub before leaving your desk. Write some notes for each commit to document what has happened.

### Step 4. Compare Code

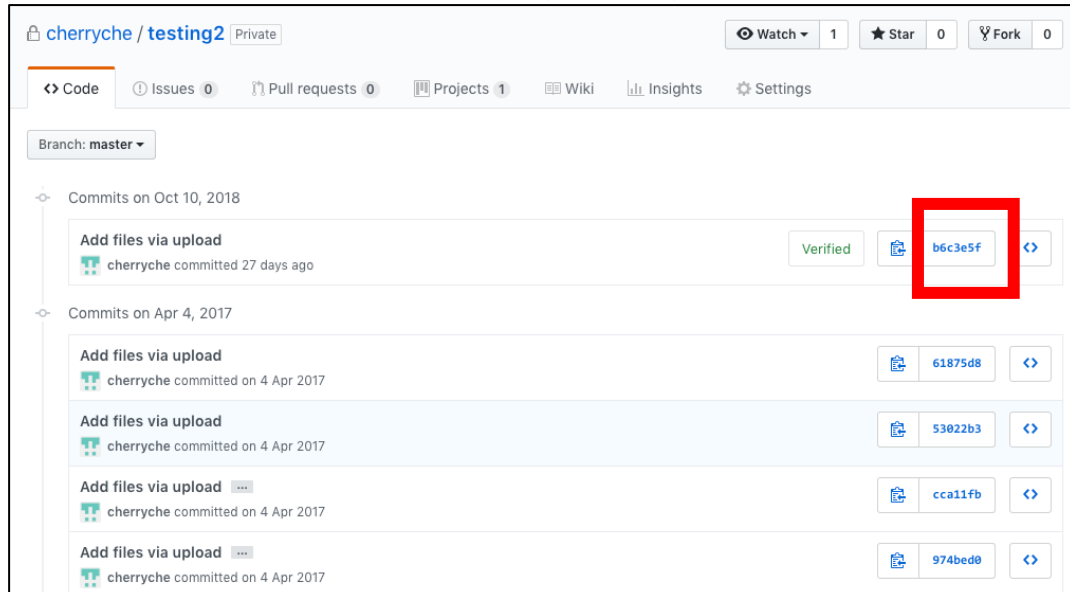
Make some changes to your software files that you uploaded earlier. It does not matter what changes you make – just alter something so as to learn how to compare differences between two sets of code.

Follow these steps:

1. Repeat Step 3 to upload the newly changed software project (or files) to your repository.
2. You may see “X commits” in the main page, click it.



3. You can now see a list of commits in the history, e.g. something like:



4. Each commit is associated with a unique ID. Click the commit or the ID, and you can see the difference between this commit and the one before. The default view is a split view. Example:



## Step 5. Fork

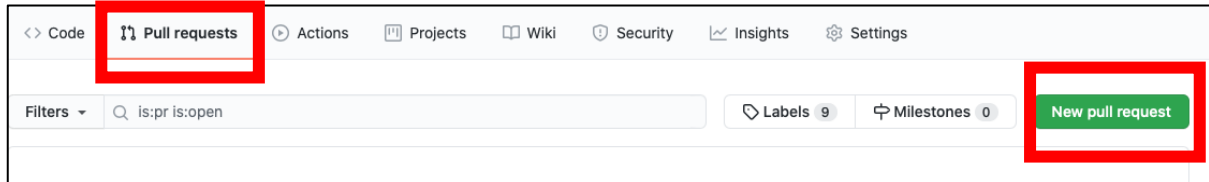
For the purpose of experiment, you can access the repository as shown below:

[https://github.com/cherryche/testing\\_public](https://github.com/cherryche/testing_public)

Fork it to your own repository. Now make some changes and commit.

## Step 5. Pull Request

Navigate to “Pull requests” tab and click “New pull request”.



Write some comments so the original owner could see your intention.

Then just await the approval from the original contributor.

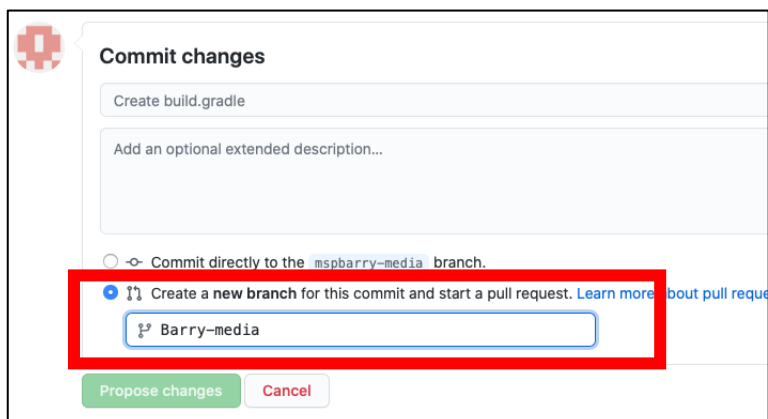
## Step 6. New Branch

In your own repository, make some changes to the code (again, for the purpose of exercise, any random change is fine). But this time, let's learn a new way of making changes.

GitHub allows you to edit code directly from the web interface. Simply locate the file you wish to edit and hit the edit button (as shown below).



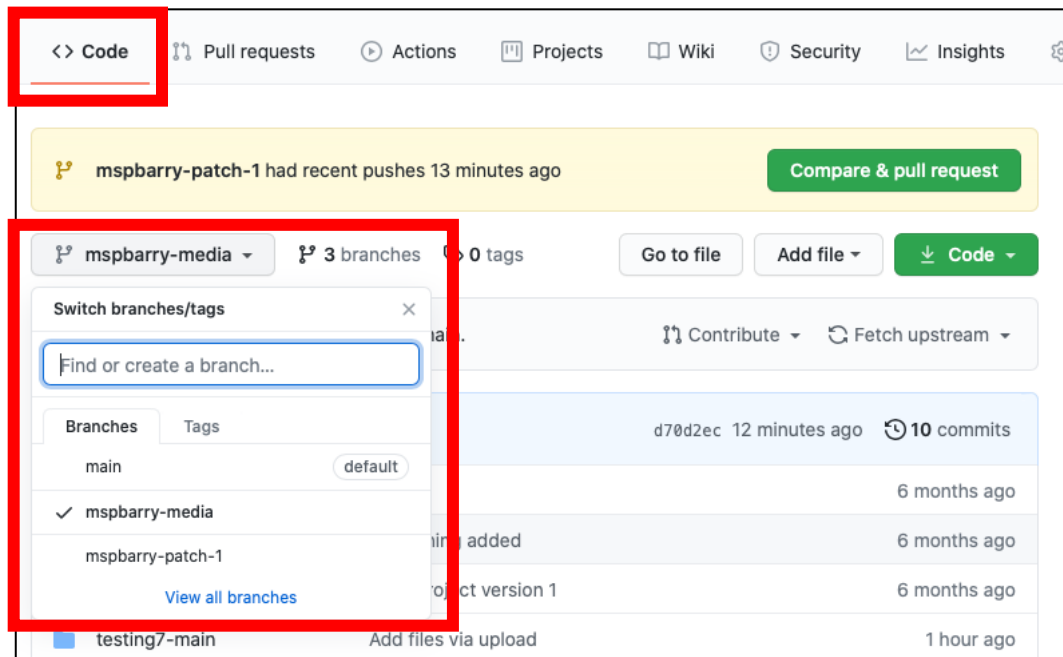
When finish editing, choose “Create a new branch...” option to commit. Give a meaningful name.



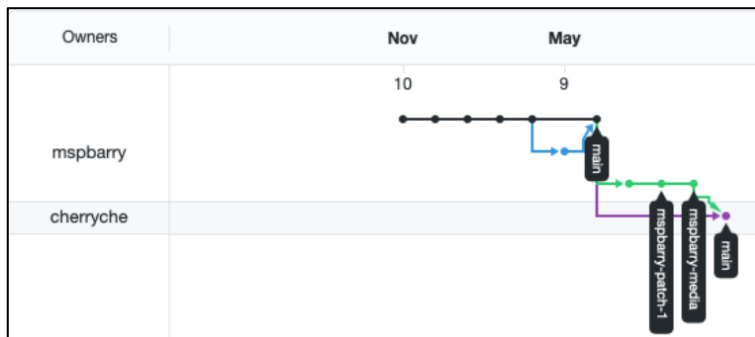
Once you click “Propose changes”, GitHub assumes you would like to do a Pull Request. You can ignore that for now.

In the future, when you browse or change files, you need to make sure you are dealing with the right branch.

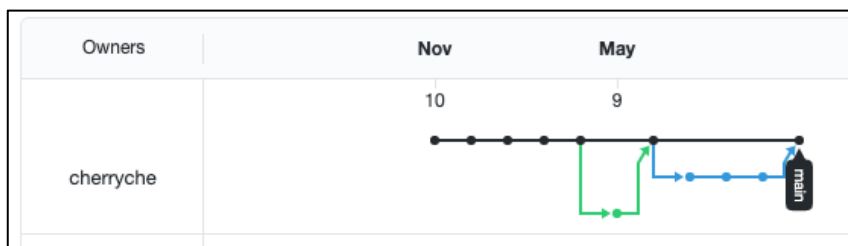
Click “Code” → the name of your current branch. There is a dropdown menu for you to navigate through different branches.



If you click “Insight” → “Network”, you may have a view of your branch activities. It may look something like this (yours may vary):



Should you wish to, you can create Pull Requests to the original owner (make sure to choose the correct branch for the Pull Request). If your Pull Request is approved and merged, they may have a view with something like this:



You may need to solve **merge conflict** if any. Do some research or ask generative AI for help if there are problems you cannot solve.

## Frequently Asked Questions

### What should I do about the 50MB file size limit during committing?

GitHub gives a warning of 50MB if the file upload exceeds 50MB for the commit. There is a 100MB hard restriction.

Typically, 50MB should be enough for most common coursework of an undergraduate module, e.g. medium Java project is about 50,000 - 200,000 lines (5MB - 20MB), and GUI-heavy Java project may be 250,000 - 300,000 lines (50MB).

In the unlikely event that your project files exceed the given limit, you can use a hybrid approach:

- Store **core project files** in GitHub
- Store large files in a cloud such as Google Drive / Dropbox / AWS S3 (such as images, audio files)
- Use scripts to download the files when needed (ask ChatGPT or DeepSeek for detailed instructions)

### How to commit to GitHub if my allocated task is about data collection or software testing?

You can still make a commit to the project repository on GitHub even if they are not writing core code, which will function as important evidence of your contribution. For example:

- **Commit collected data** or **commit metadata/scripts** related to data collection. (Refer to the above if file size is too large.)
- **Add test cases** and commit test scripts (e.g., unit tests, integration tests).
- **Update documentation** (README, test plans, data collection notes).

### What happens to project files when merging team members' feature branch to the master branch?

GitHub applies file-by-file merging, meaning it integrates the differences within the same directory structure.

For example, when merging main code (Y) with testing scripts (E) in a Python project,

```
main: A → B → C → X → Y → M (merge commit)
      |
      +-- D → E
```

#### Before Merge:

- Branch main (Y - Main Code)

```
/project
├─ app.py
├─ config.yaml
└─ utils/
   └─ helper.py
      └─ formatter.py
```

- Branch feature (E - Testing Scripts)

```
/project
├─ app.py (modified)
├─ tests/
   └─ test_app.py
      └─ test_helper.py
```

**After Merge:**

- Merged main (now contains Y + E)

```
/project
├─ app.py (merged version)
├─ config.yaml
├─ utils/
│   └─ helper.py
│   └─ formatter.py
├─ tests/ (new folder added from feature branch)
│   └─ test_app.py
│   └─ test_helper.py
```