School of Computing & Communications | Lancaster University

SCC.111 Software Development – Lecture 32: Collaborative Workflows

Adrian Friday, Hansi Hettiarachchi and Nigel Davies

Introduction

Lancaster University

- In the last few lectures, we looked at:
  - Component organisation in Java using **Layout Managers**
  - A real examples of **composition** in action
  - Asynchronous programming

- Today we're going to investigate how OO design also helps to promote efficient and safe collaborative software development…

- **We'll also see how version control tools can help to facilitate this**

Version Control Refresher…

Lancaster University

In term 1 we saw how a version control tool called git helps to provide resilience

- Git repositories (repos) are typically stored on a server somewhere
- Repos are then **cloned** onto other computers as needed (e.g. your laptop)
- All changes made within a repo are automatically tracked.
- When we have made some useful change (fix a bug, add a new feature…), we **commit**.
- As frequently as we like we **push** those commits back to the server
- As frequently as we like we **pull** changes into our local repo from the server
- All versions of your code are stored, so no work is ever truly lost

Today we're going to see that in action.

- We are going to collaboratively create some software using the GameArena classes from this week's lab. **Watch and learn.** ☺

Creating a Repository

The simplest way is to do this interactively via your favourite provider…

- GitHub and GitLab are some of the most popular
- We will use GitLab today ☺

- Sign up if you haven't already done so.

https://scc-source.lancs.ac.uk/

Creating a Repository

Lancaster University

Your work / Projects

Search or go to...

Your work

- Projects
- Groups
- Issues
- Merge requests
- To-Do List
- Milestones
- Snippets
- Activity

Help

Projects

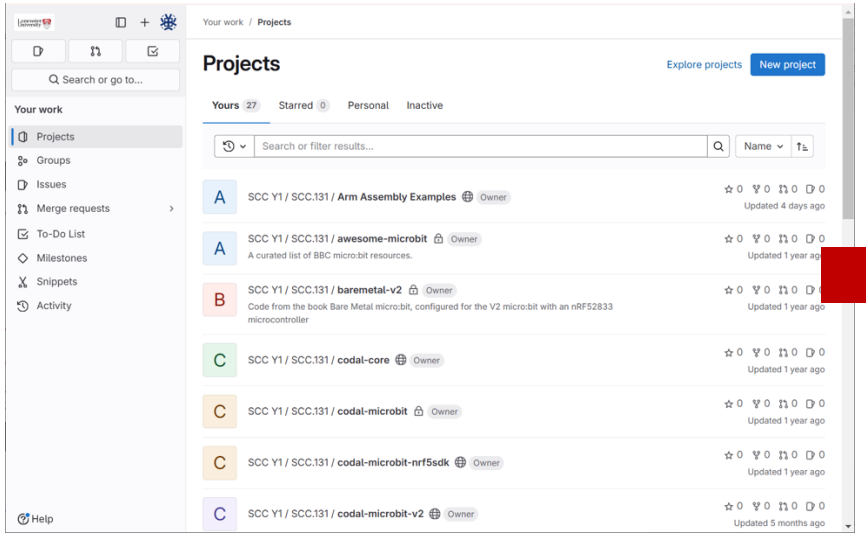
Explore projects | New project

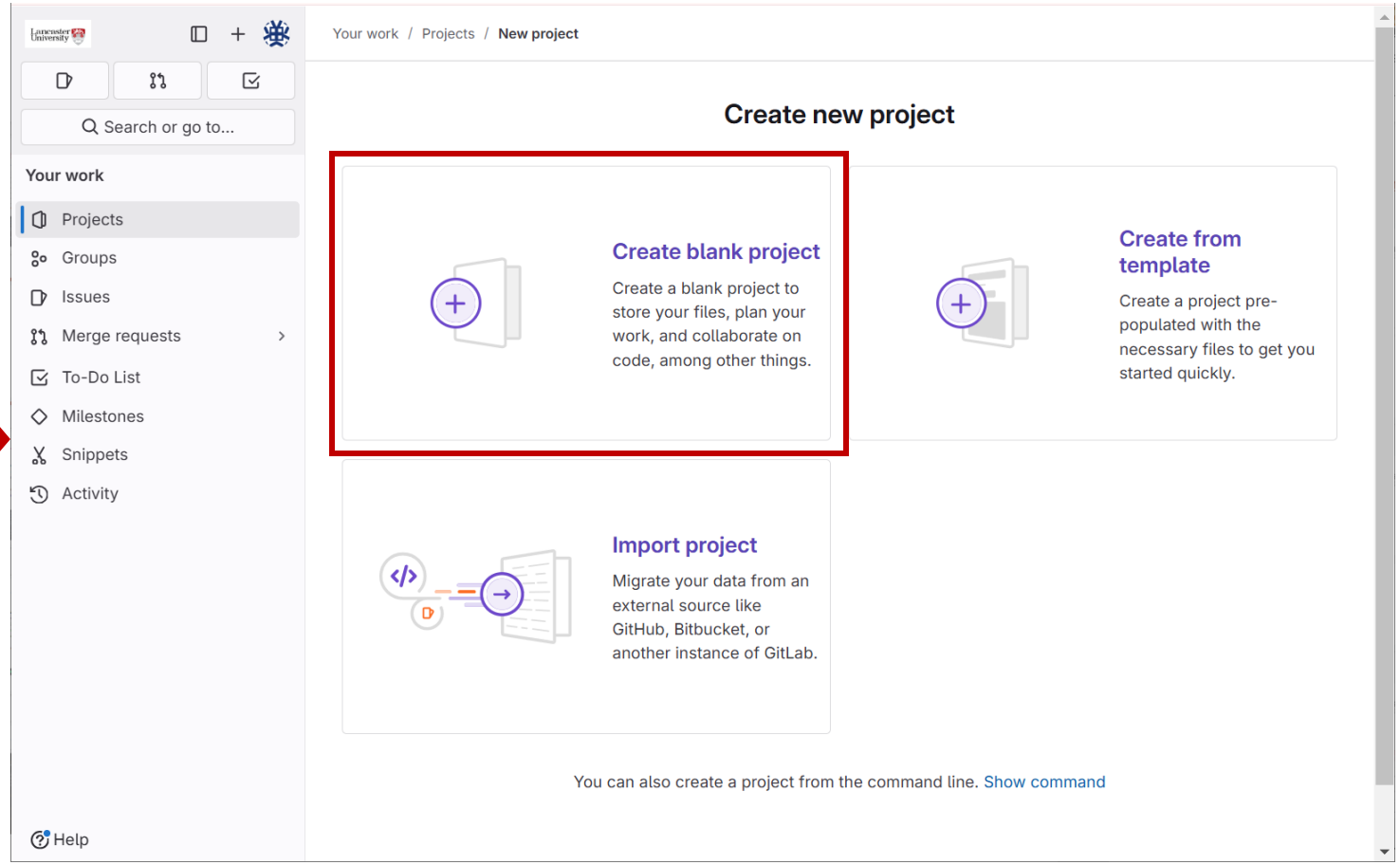
Yours 27 | Starred 0 | Personal | Inactive

Search or filter results... | Name

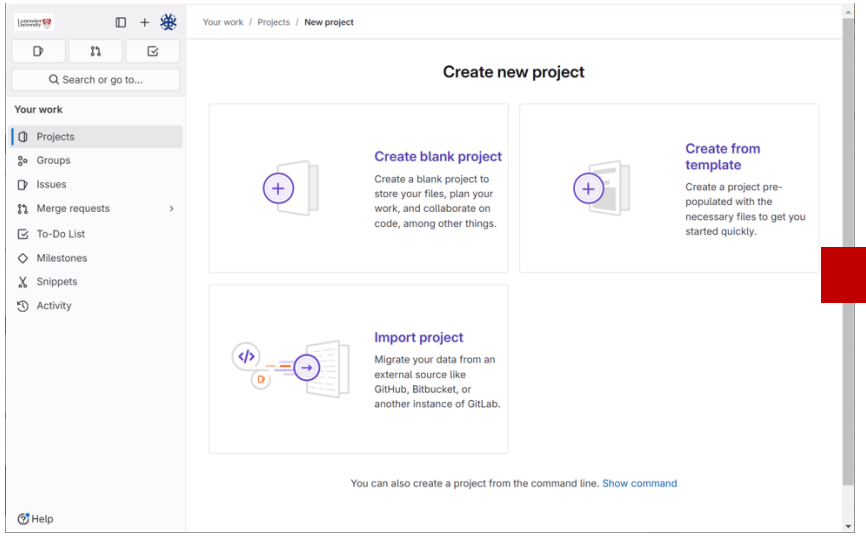
- A — SCC Y1 / SCC.131 / **Arm Assembly Examples** Owner — Updated 4 days ago
- A — SCC Y1 / SCC.131 / **awesome-microbit** Owner — A curated list of BBC micro:bit resources. — Updated 1 year ago
- B — SCC Y1 / SCC.131 / **baremetal-v2** Owner — Code from the book Bare Metal micro:bit, configured for the V2 micro:bit with an nRF52833 microcontroller — Updated 1 year ago
- C — SCC Y1 / SCC.131 / **codal-core** Owner — Updated 1 year ago
- C — SCC Y1 / SCC.131 / **codal-microbit** Owner — Updated 1 year ago
- C — SCC Y1 / SCC.131 / **codal-microbit-nrf5sdk** Owner — Updated 1 year ago
- C — SCC Y1 / SCC.131 / **codal-microbit-v2** Owner — Updated 5 months ago

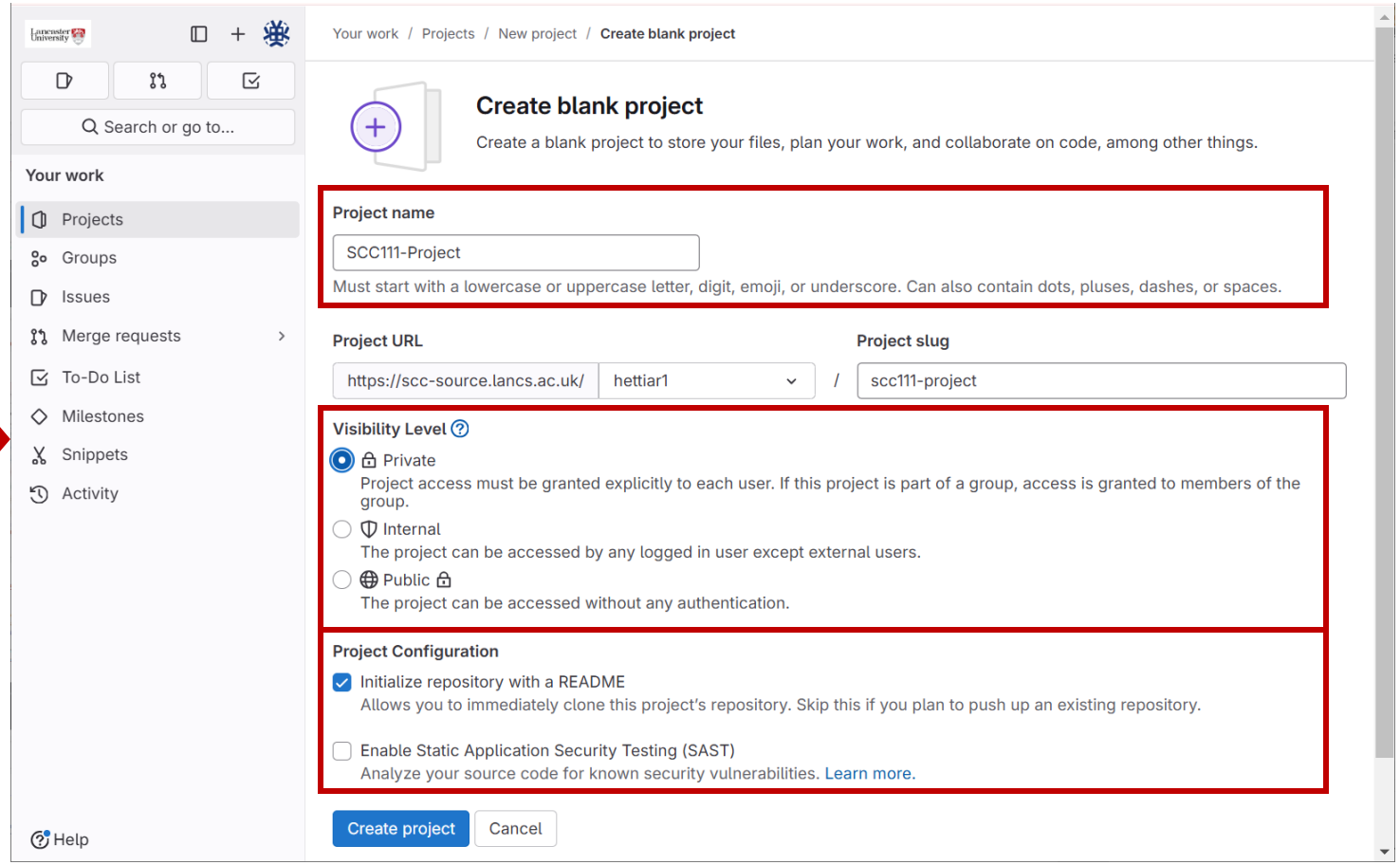
Creating a Repository



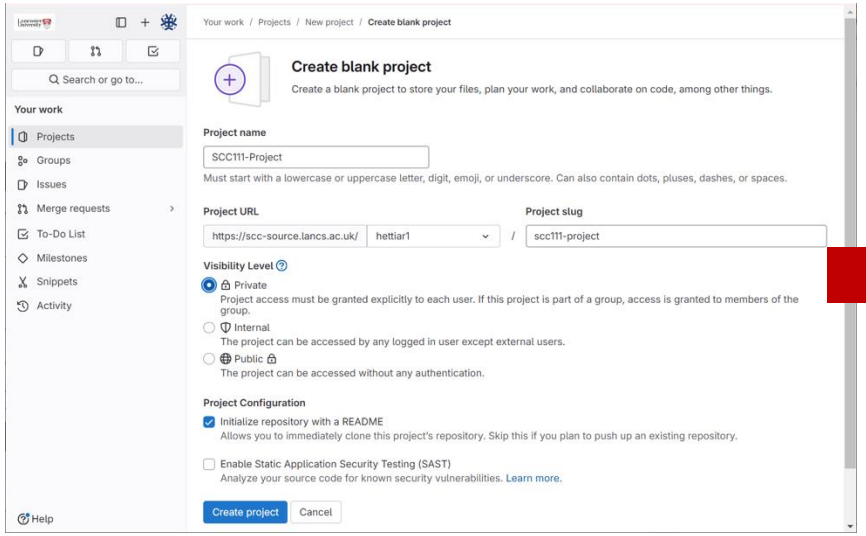


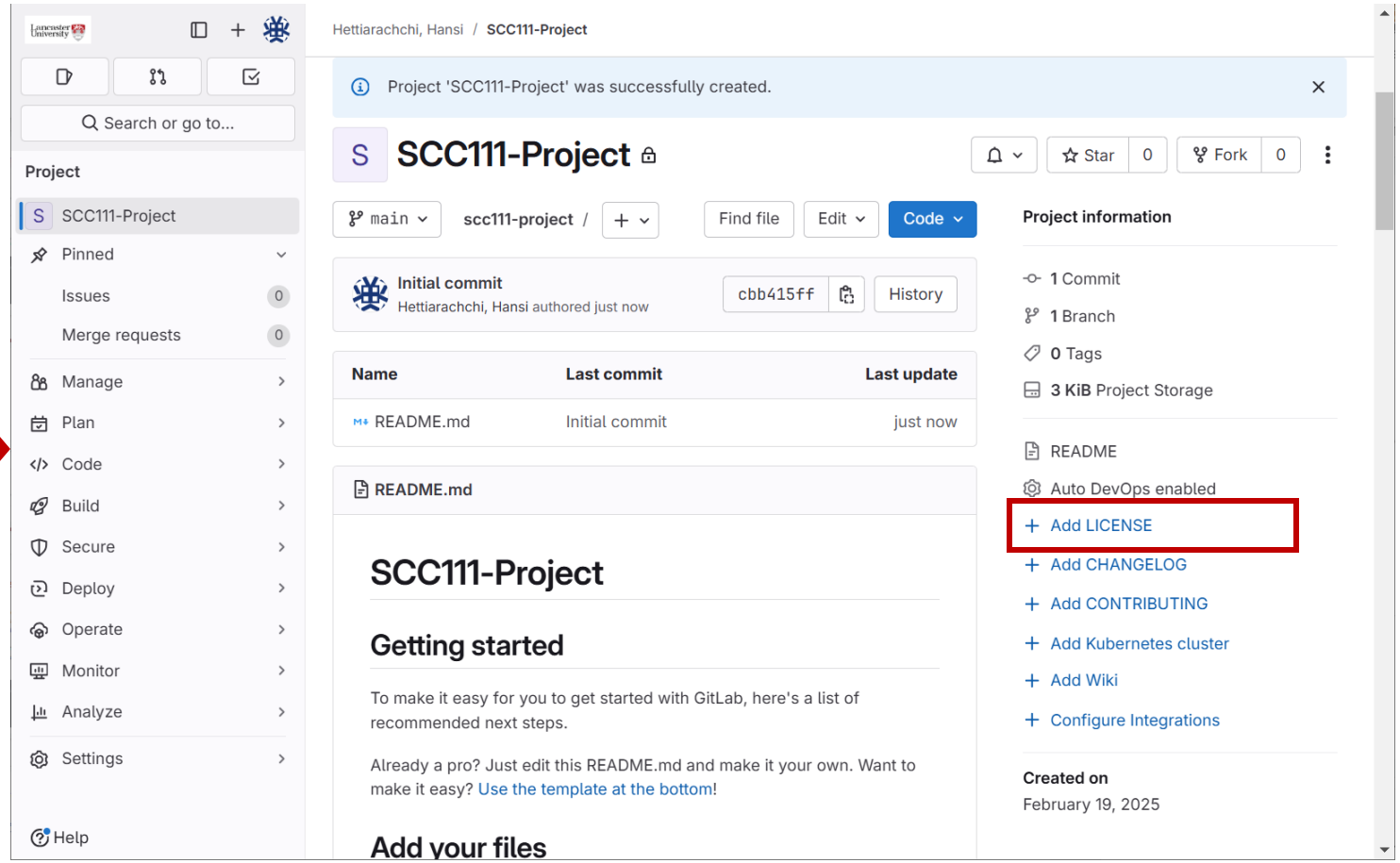
Creating a Repository



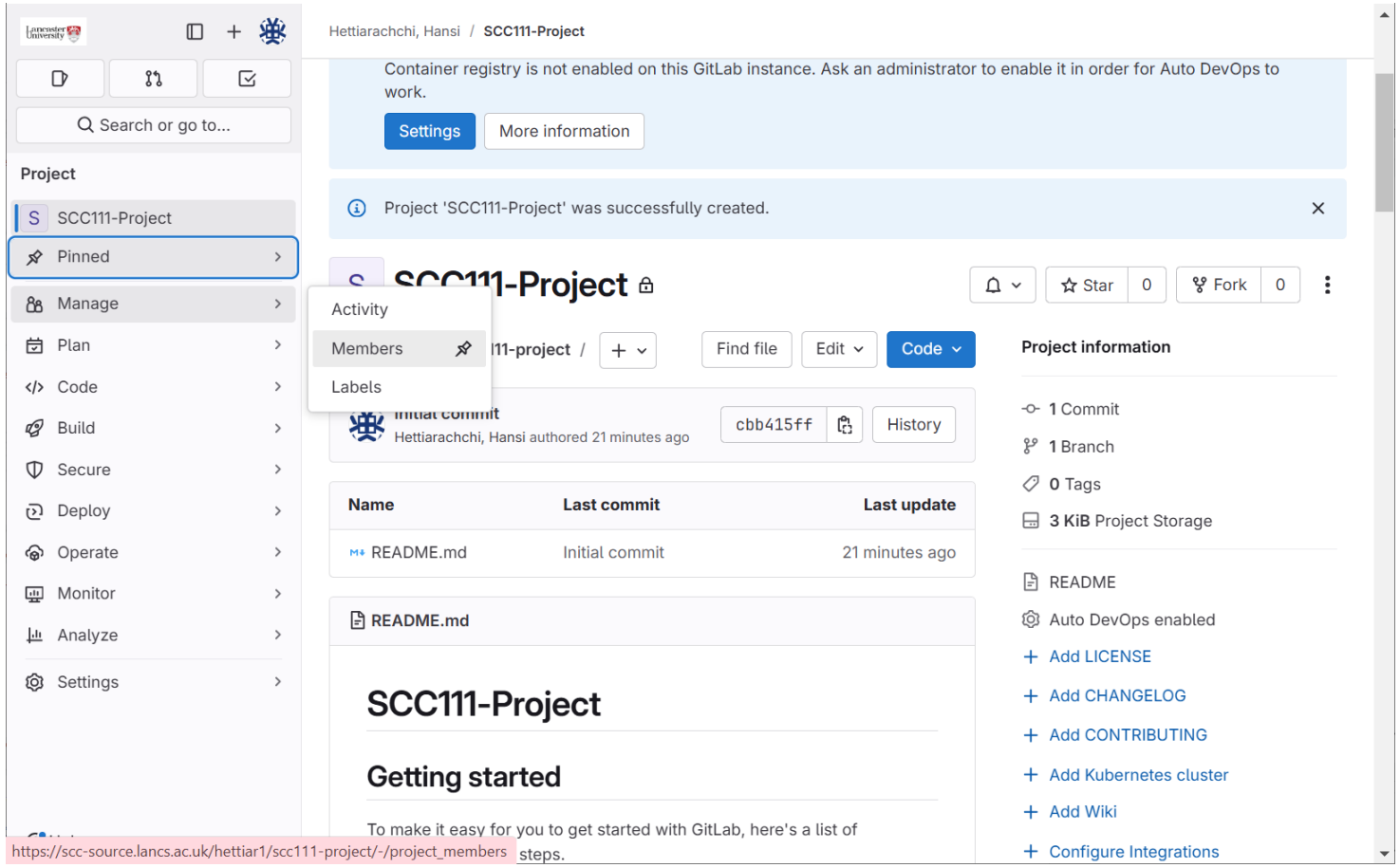


Creating a Repository

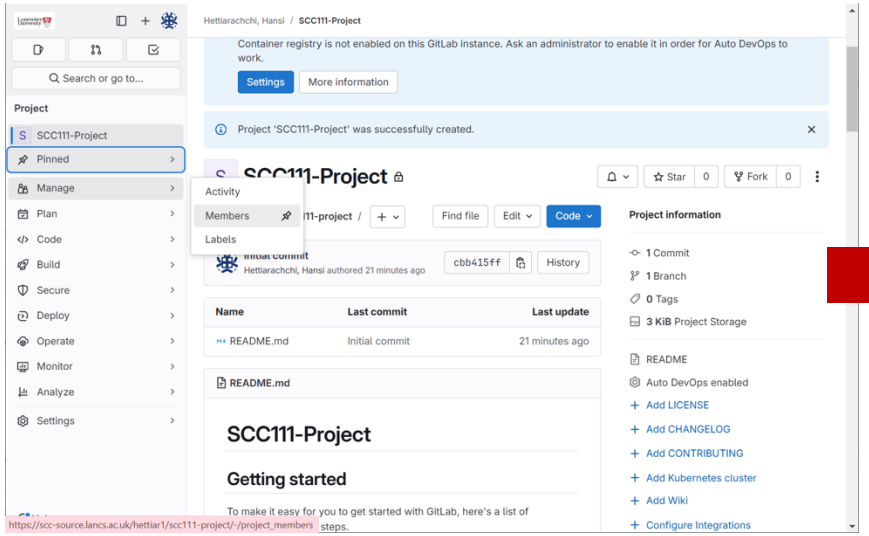


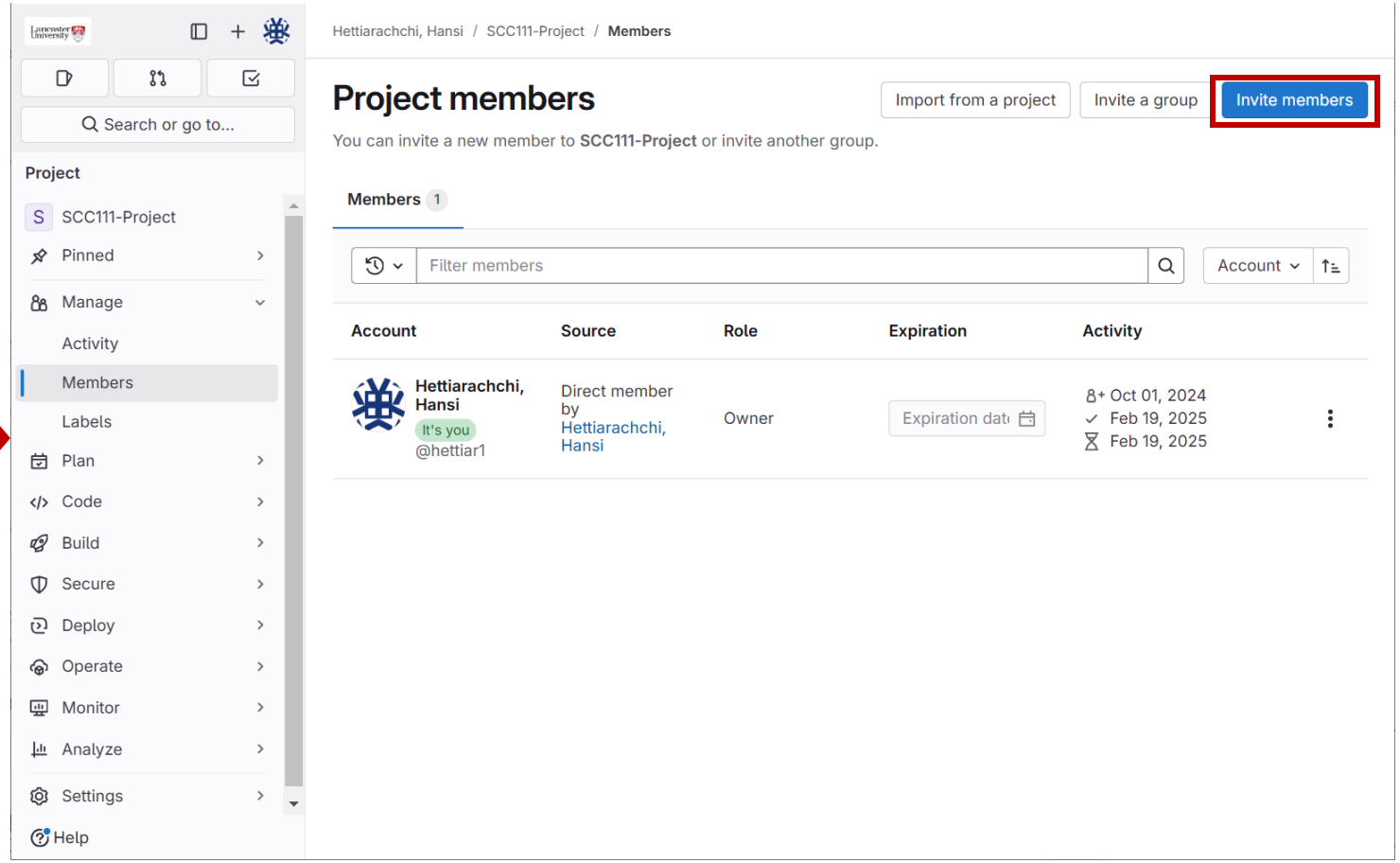


Adding Contributors/Members

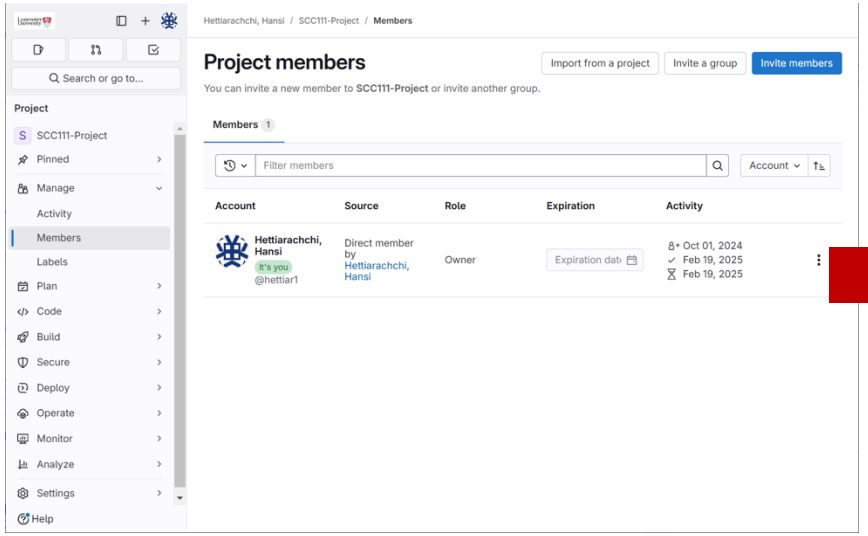


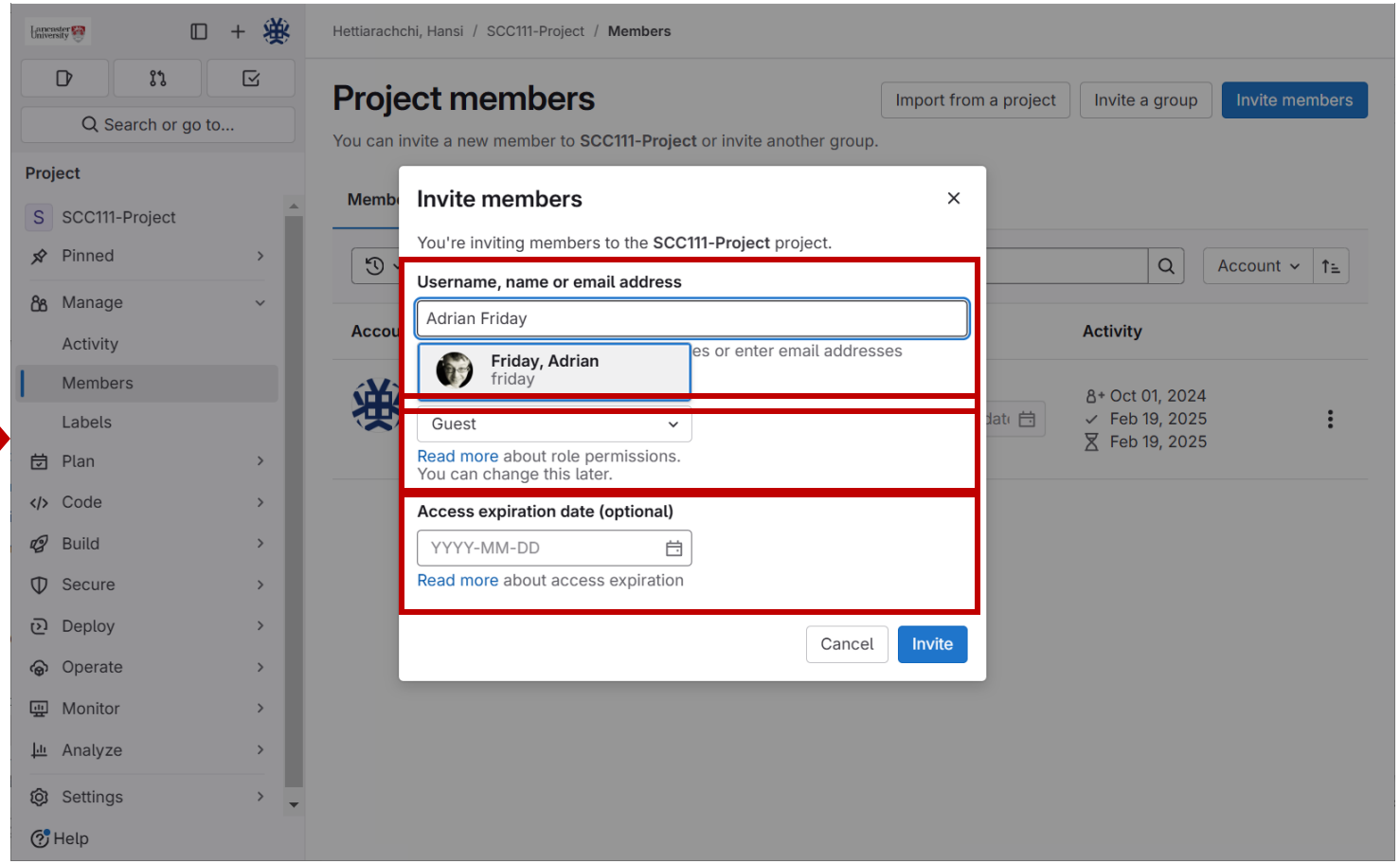
Adding Contributors/Members





Adding Contributors/Members





Accessing a repository for the first time

Once there is a remote repo, we often want a local copy to work on…

- There is a standard command line tool called git.
- There are also GUI tools, and VS Code integration.
- The command line tool provides more control over your work, so is good to learn.
- **We recommend you learn the command line tool in this course.**

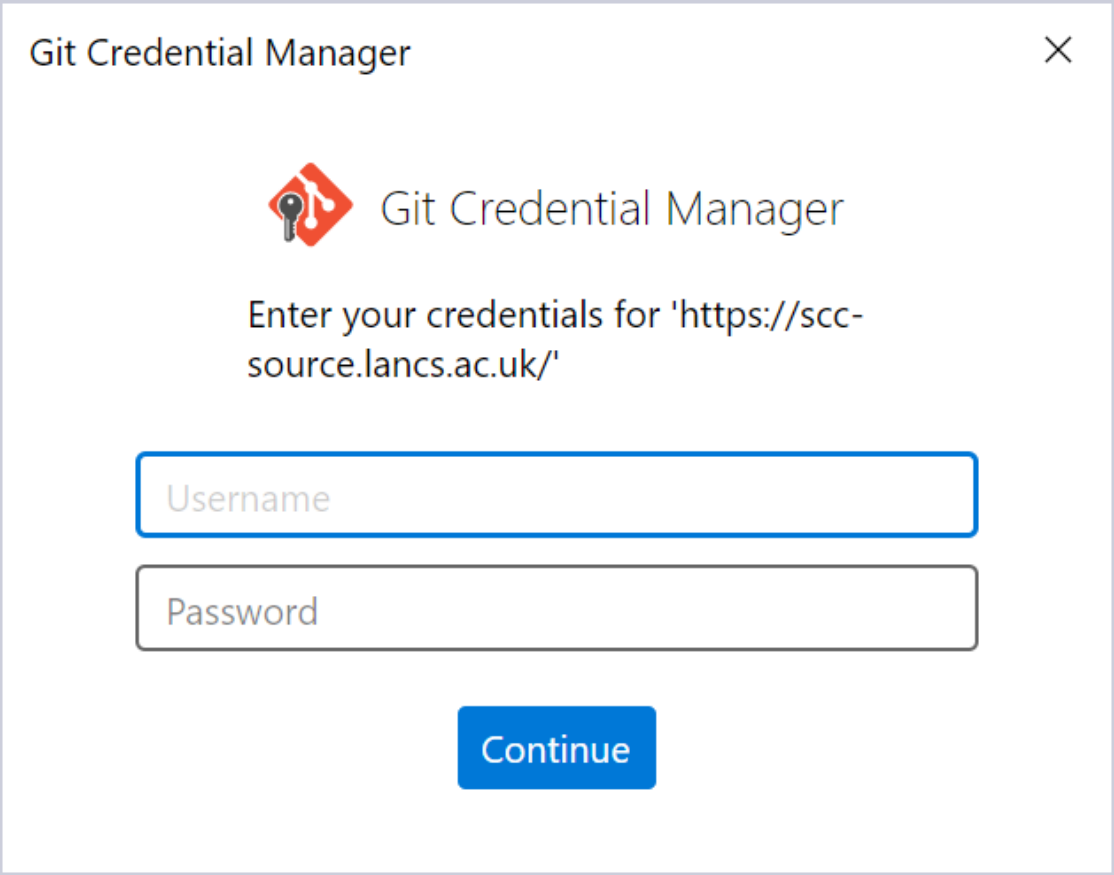
- To take a local copy of a git repo, we use the **git clone** command:

```
git clone https://scc-source.lancs.ac.uk/hettiar1/scc111-project.git
```

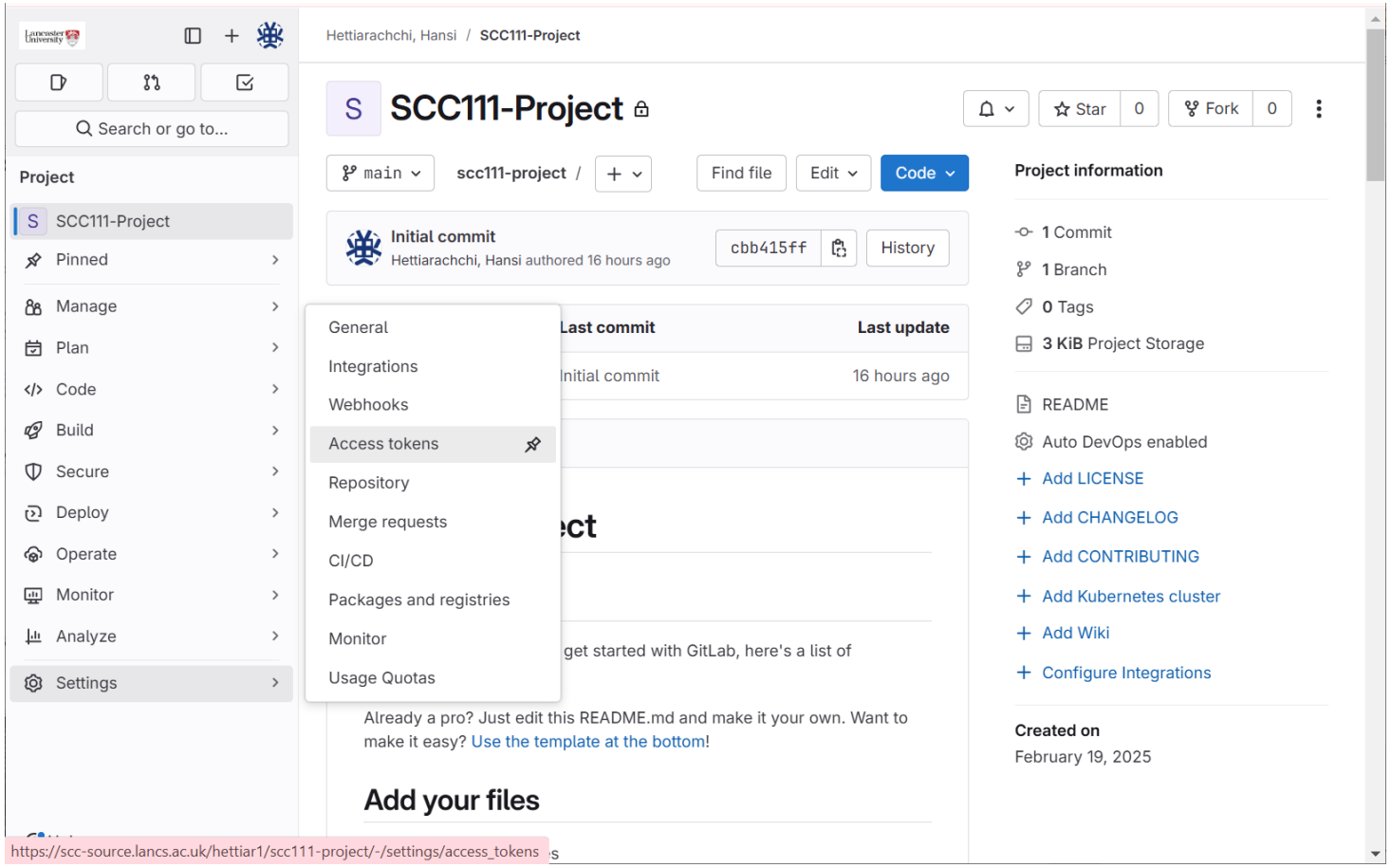
Accessing a private repository

- If your repository is private, you need to provide your credentials.
- It is recommended providing an access token as the password.

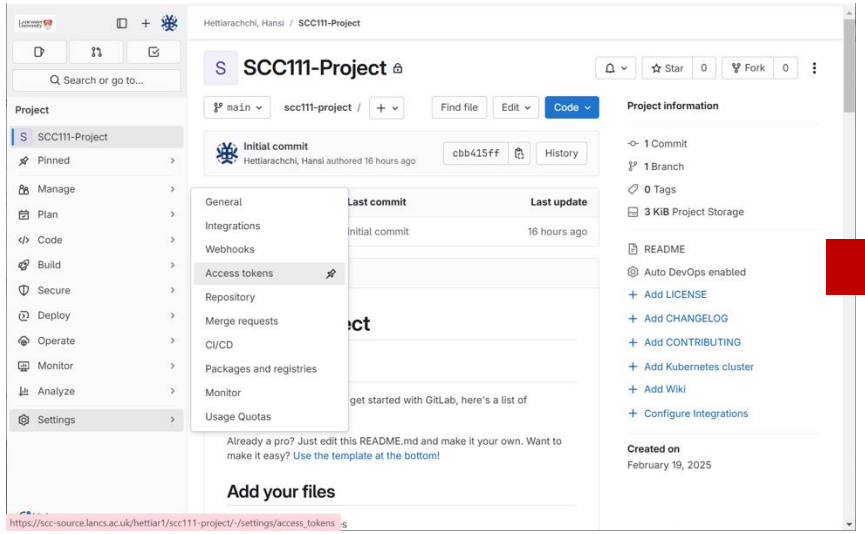
- Access tokens are similar to passwords, except you can limit access to resources, select a limited role, and provide an expiry date.

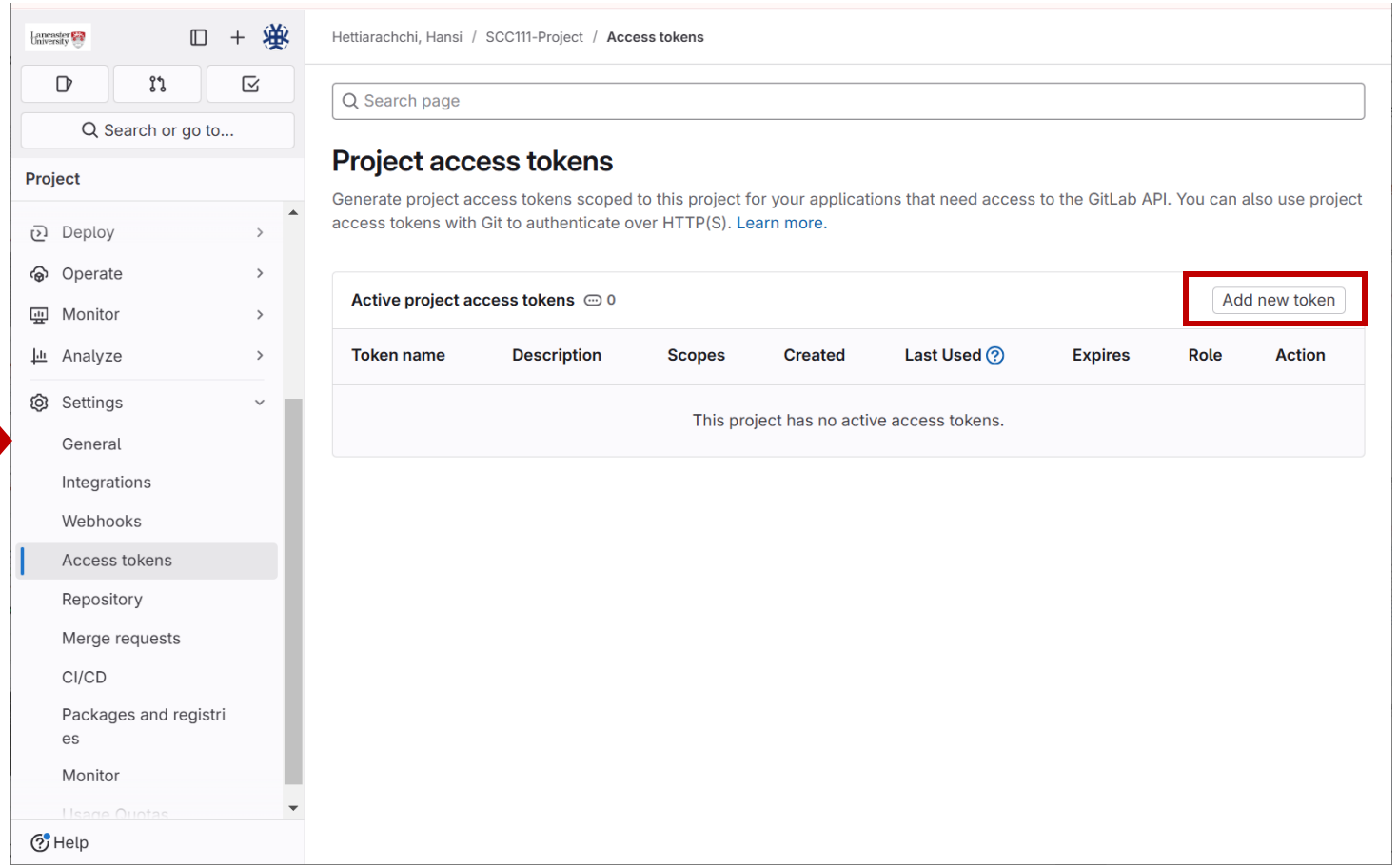


Creating a project access token

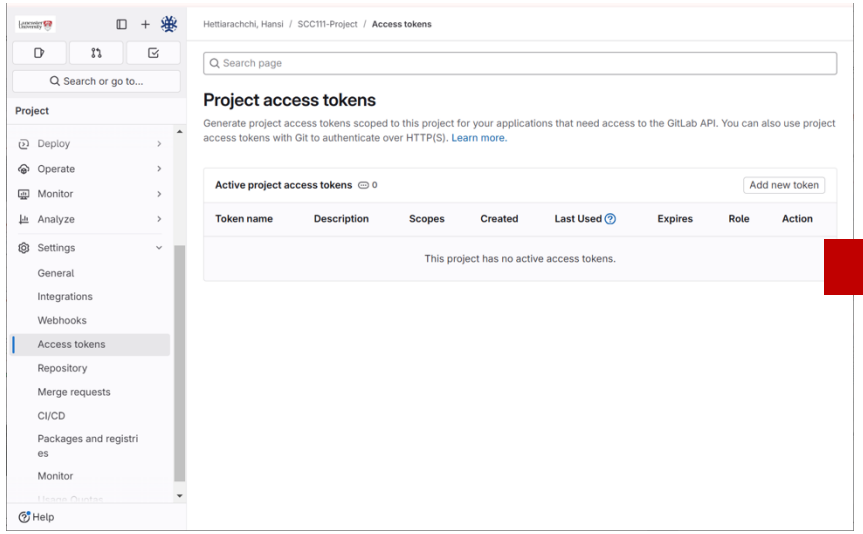


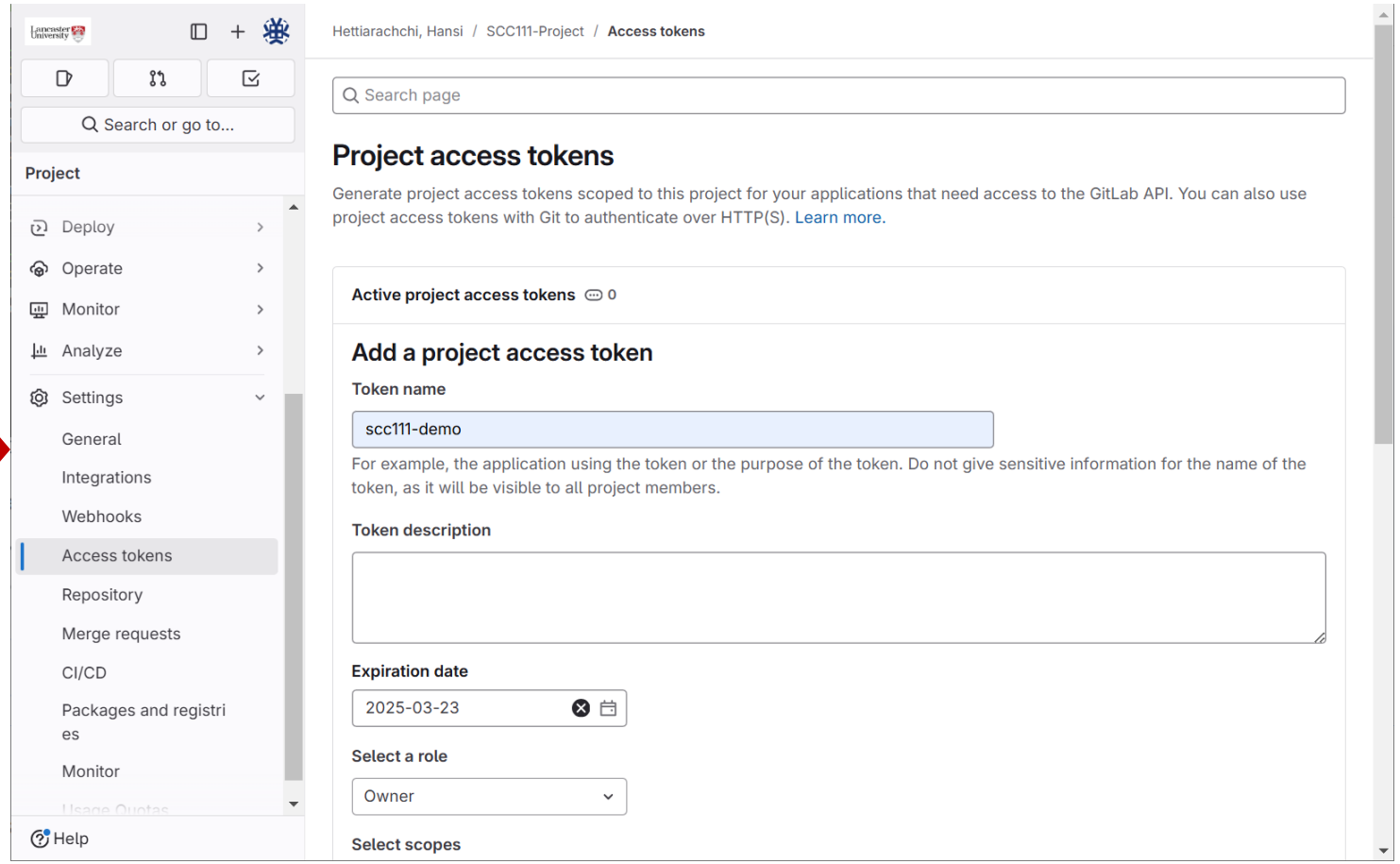
Creating a project access token





Creating a project access token





Accessing a repository for the first time

MINGW64:/c/Users/hettiar1/OneDrive - Lancaster University/Teaching/SCC111/Co...

```
hettiar1@LU-GM0K1F09 MINGW64 /c/Users/hettiar1/OneDrive - Lancaster University/T
eaching/SCC111/CodeSamples/L32
$ git clone https://scc-source.lancs.ac.uk/hettiar1/scc111-project.git
Cloning into 'scc111-project'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Receiving objects: 100% (3/3), done.

hettiar1@LU-GM0K1F09 MINGW64 /c/Users/hettiar1/OneDrive - Lancaster University/T
eaching/SCC111/CodeSamples/L32
$ cd scc111-project

hettiar1@LU-GM0K1F09 MINGW64 /c/Users/hettiar1/OneDrive - Lancaster University/T
eaching/SCC111/CodeSamples/L32/scc111-project (main)
$ ls
README.md

hettiar1@LU-GM0K1F09 MINGW64 /c/Users/hettiar1/OneDrive - Lancaster University/T
eaching/SCC111/CodeSamples/L32/scc111-project (main)
$ |
```

What Git does…

Once you have a local repo, your computer is tracking ALL changes you make in that folder.

- New files
- Deleted files
- Changed files

Your local copy will not change by itself – even if someone else updates the remote repo

- You can trust the local files no to change
- You can trust that **ANY** changes you make locally will not irrevocably affect others
- You can trust that any code that has been committed can never be truly destroyed as long as the repo exists.

Adding new files…

If you create a new file in a repo, this will be noticed

- But it will not automatically become part of the repo
- Do this explicitly
- Once added, the files content will be tracked for changes.

- **Therefore, only add files you want to be tracked.**
- Use "git status" to show untracked files, and changes in tracked files.
- Use "git add" to add a newly created file to a repo

```
git status
```

```
git add <filename>
```

Committing changes…

When you reach a good "checkpoint" commit your changes

- E.g. fixed a bug, added a new feature…
- Keep commits clean (one meaningful code change per commit)
- Choose a name for a commit that describes the change
- Commit often and include as much detail as necessary.

- **Review changes before you commit. Did you intend to make all those changes??**

```
git add <filename>
```

```
git commit -m "A short description of what you changed."
```

Sharing changes…

Once you have one or more local commits, you can share back to the server.

- This makes your changes visible to others
- It also means your work is properly backed up…
- Until you push, you are still vulnerable to local loss hardware fail, theft, damage, accidental deletion…

Use git push to propagate your commits back to the server.

- We specify where to push to (normally "origin" – wherever you cloned from) and the branch to push (the "main" branch is fine for now!)

```
git push origin main
```

Receiving changes…

You can receive updates from others via explicitly merging them into your local repo

- Changes only arrive when you want them to.
- Use **git pull** to do this
- Any commit you don't have are then downloaded and merged into your local repo.

Unless both you and someone else have editing the same file since you last pulled, no user intervention is needed…

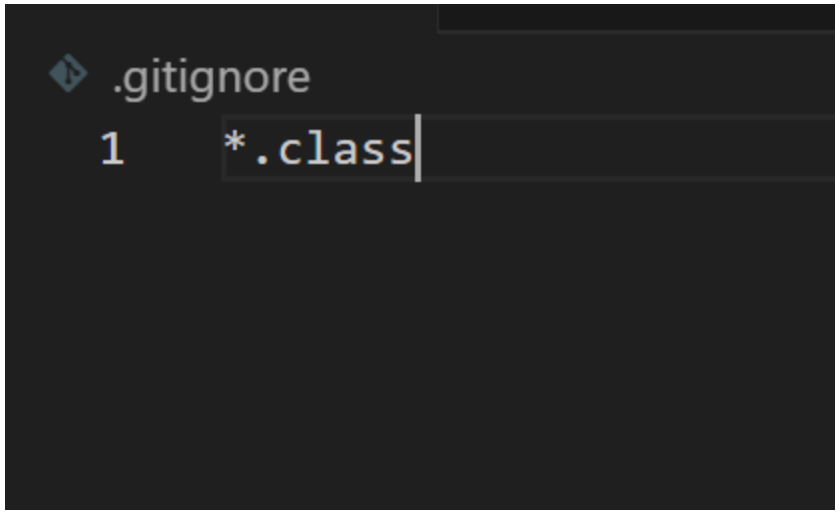
If the version on the server is newer than your local version, you will be required to perform a pull operation before you can push.

```
git pull
```

Ignoring changes…

- **We might need to keep some files only in our local repository (e.g. .class files).**
- You can configure Git to ignore such files.

- Create a **.gitignore** file in your local repository's root directory to tell Git which files and directories to ignore when you make a commit.



Merge conflicts…

If someone else has updated file on the server as you have edited locally…

- This is called a merge conflict
- Git will need some help from you to resolve this.

- Any files in conflict will be updated with **BOTH** copies of lines of code that differ
- Fix this conflict by ether choosing the versions of the lines you want
- Or by editing the code until it is correct.

- Then add these changed files that were in conflict and commit.

```
git add
git commit
```

Summary

Today we learned:

- How version control can promote collaboration
- Practical guidance on using Git version control
- Examples of a typical Git Workflow

Some open-source repos you might find interesting:

https://github.com/lancaster-university/infolab-lights/

https://github.com/lancaster-university/microbit-v2-samples

https://github.com/openjdk/jdk

https://github.com/finneyj/GameArena