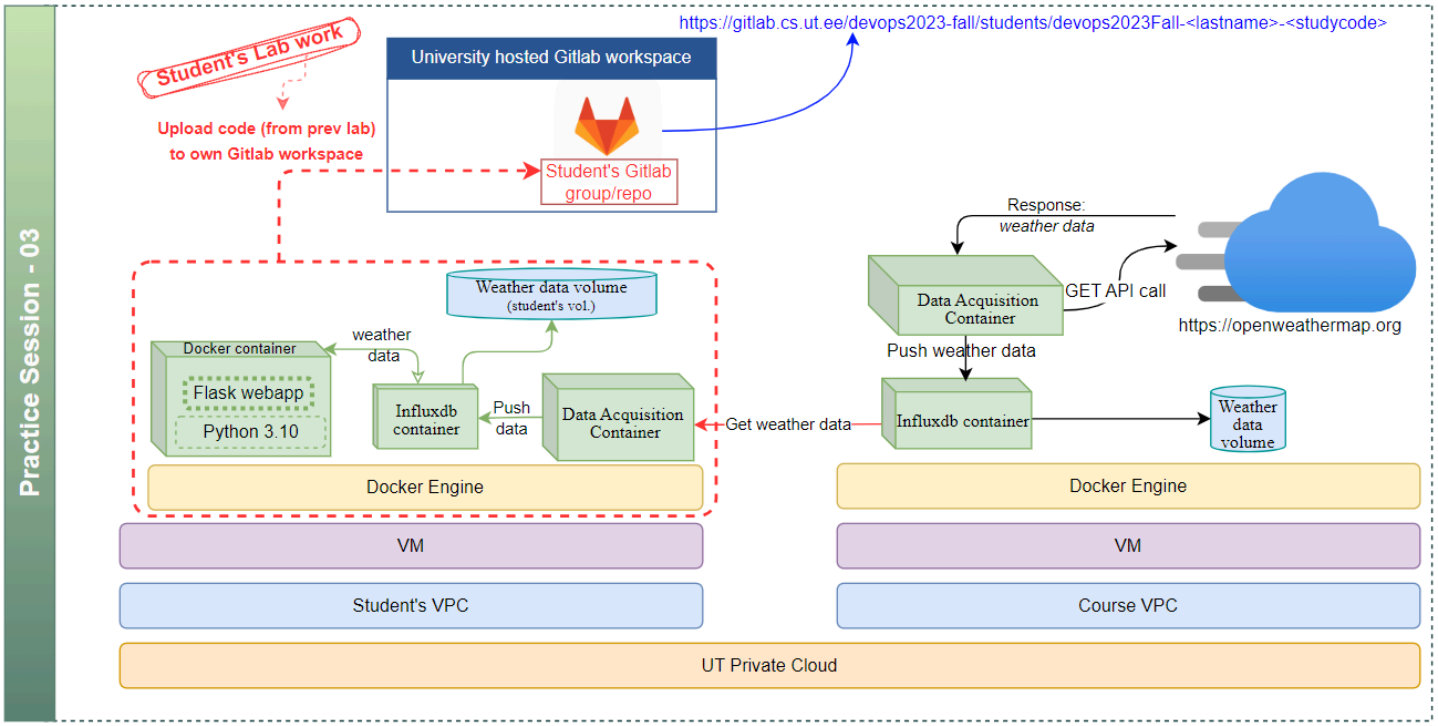
# Practice Session 3: Git: Version Control System

**Make sure that you have already gone through Lab-02**

In this practice session, you will get familiar with the `git` version control system. You will use the Institute's GitLab environment. After this practice session, you should be able to

- create and manage the repository
- create and merge branches
- resolving merge conflicts
- Getting involved with other projects through forking the repo. creating merge request, etc
- you should be able perform most of the above tasks through command line interface

## What are we going to do?




During this lab session, your tasks will involve familiarising yourself with fundamental Git commands up to Exercise 4. In the deliverable section, you will be required to create a GitLab project for each of the previous lab materials and subsequently push the corresponding code and associated screenshots to their respective projects as shown in the above Figure. As an illustration, for Lab01, you should push the "flask-app" directory, along with any screenshots captured during the Lab01 practice session.

# Pre-Requisites:

basic bash commands; `ls`, `vim` or `vi`, `cat`, `mkdir`, `cd` etc.

**Please Note!!**

We are performing this lab on the Linux terminal (bash) of controller VM, however you can also try with the following IDEs for example Visual Studio Code, Git Desktop.

- Visual studio code has several features for example, ssh to vm, create code, use git etc.
  - Running on Windows, Linux, Mac
- Git Desktop is basically used to manage your code in VCS (github or gitlab)
  - Running on Windows, Linux, Mac

## Exercise 1: Creating first repository

**Intro**: In this exercise you will get familiar with the basic Git commands, including installation and configuration of `git`, creating repositories, cloning repositories etc. You will not use Github rather the GitLab environment provided by the institute.

### 1.1: Create Institute GitLab environment

In this practice session, we will not use GitHub. We will use the Institute provided GitLab environment.

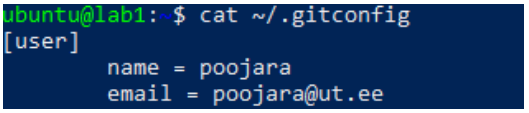
1. Go to https://gitlab.cs.ut.ee/
2. Login using University `username` and `password`

### 1.2: Connecting to virtual machine and check for *git* installation

1. SSH to your VM
2. Check for installation `git --version`

### 1.3: Configure Git

1. To review the configuration setting at any time, issue the following command.
   `git config --list`
2. For global settings you may use `--global` option, e.g. `git config --global --list`
3. Configure user information to be used for all the local repositories. Make sure you put your information in quotes " ".
   - `git config --global user.name "Your Name"`
   - `git config --global user.email "your@email.id"`
4. you can find the configuration file at `~/.gitconfig`
   - To see the configuration file enter `cat ~/.gitconfig`




### 1.4: Create first repo locally

1. Open Git Bash Terminal
2. Goto your home directory
3. Create a directory with the name you want to use as your repository name.
    - `mkdir firstrepo`
    - `cd firstrepo`
4. Initialise the git repo with
    - `git init` command
    - This will add a `.git` directory with some necessary information
    - You can go inside and check the directory

## 1.5: Status of the repository

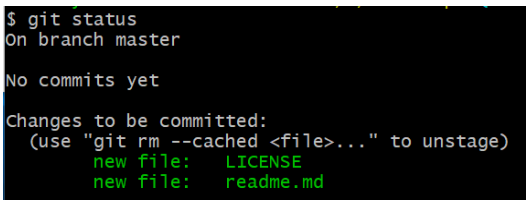
1. This refers to the state of the working directory and the staging area. Enter the command `git status` inside firstrepo directory.

```
On branch master
No commits yet
nothing to commit (create/copy files and use "git add" to track)
```

`DIY: learn by yourself the meaning of the above lines. You should be able to recall the concepts of branching, commit, tracking/staging.`

## 1.6: Staging the file

1. What is staging? Staging is an intermediate phase prior to committing a file to the repository with the `git commit` command.
2. Now you are inside `firstrepo` directory.
3. Let's first create two new empty files.
    - `touch LICENCE`
    - `touch readme.md`
4. Now if you enter the `git status` command, you should be able to see that two files are untracked.
5. Add the above files to git tracking system. This is also referred to as *staging the files*.
    - `git add LICENCE`
    - `git add readme.md`
6. Now check the status of your repo using the `git status` command and find the meaning of the output by yourself.
    - `git status`




## 1.7: Committing the files

1. Now you are inside `firstrepo` directory.

2. Let's make an initial commit and check the status. The `-m` option lets you give a short summary of this commit.
    - `git commit -m "Initial repo commit"`

```
$ git commit -m "Initial repo commit"
[master (root-commit) b3f6f6a] Initial repo commit
 2 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 LICENSE
 create mode 100644 readme.md
```

1. Now again check the status with the `git status` command.
2. Modifying the Existing `readme.md` file
    - Open the `readme.md` file with the `vim readme.md` command and add `"Git is cool.."` line.
    - Enter git add command: `git add .`. Here `.` (dot) at the end of the command represents everything in the current directory. In our case, the command will add only the `readme.md` file.
    - Commit the changes with `git commit` command. e.g. `git commit -m "readme file updated."`
3. Similarly, update the `LICENCE` file with the content available at https://www.apache.org/licenses/LICENSE-2.0.txt and make a commit.
4. To see the history of commits, issue `git log` command.

`DIY: Find the ans: What information can be obtained from the output of git log command?`

## 1.8: Find the difference between two changes

`git diff` command takes two inputs (e.g. hash of two commits) and reflects the differences between them. Make multiple changes and commits to the readme file.

1. Enter the `git log` command.
2. Choose any two commit hashes, say `<Commit-hash1>` and `<Commit-hash2>`
3. Enter the `git diff` command with both hashes: `git diff <Commit-hash1> <Commit-hash2>`

   `DIY: Find the ans: how to interpret git diff command?`

## 1.9: Pushing the changes to remote repo

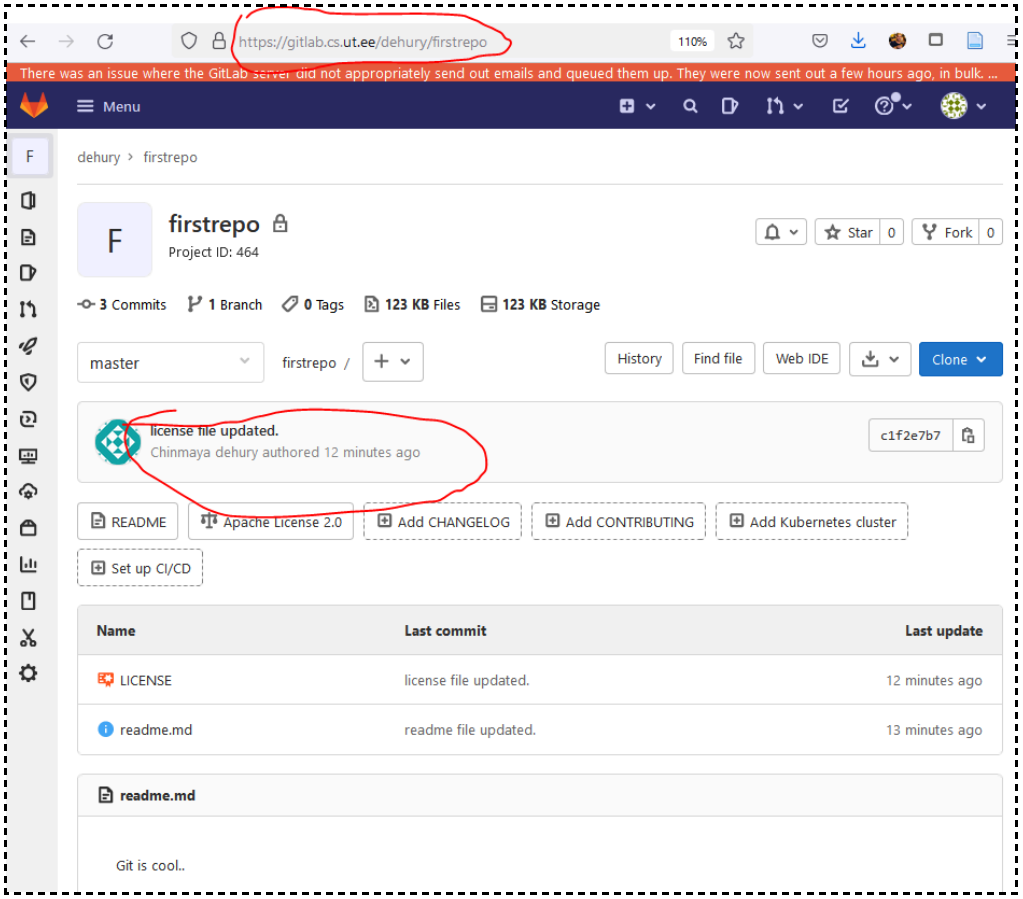
So far all the changes are made locally. Now its time to push the code changes to your remote gitlab repo using the `git push` command.

1. push the local repo to your remote gitlab account. You should replace `dehury` with your gitlab user account name.
    - `git remote add master` https://gitlab.cs.ut.ee/dehury/firstrepo
    - `git push master`

2. Screenshot 1.9-1: Take the screenshot of the output similar to below:

```
$ git push master
warning: redirecting to https://gitlab.cs.ut.ee/dehury/firstrepo.git/
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 8 threads
Compressing objects: 100% (7/7), done.
Writing objects: 100% (9/9), 4.59 KiB | 2.29 MiB/s, done.
Total 9 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote:
remote: The private project dehury/firstrepo was successfully created.
remote:
remote: To configure the remote, run:
remote:   git remote add origin https://gitlab.cs.ut.ee/dehury/firstrepo.git
remote:
remote: To view the project, visit:
remote:   https://gitlab.cs.ut.ee/dehury/firstrepo
remote:
remote:
remote:
To https://gitlab.cs.ut.ee/dehury/firstrepo
 * [new branch]      master -> master
```

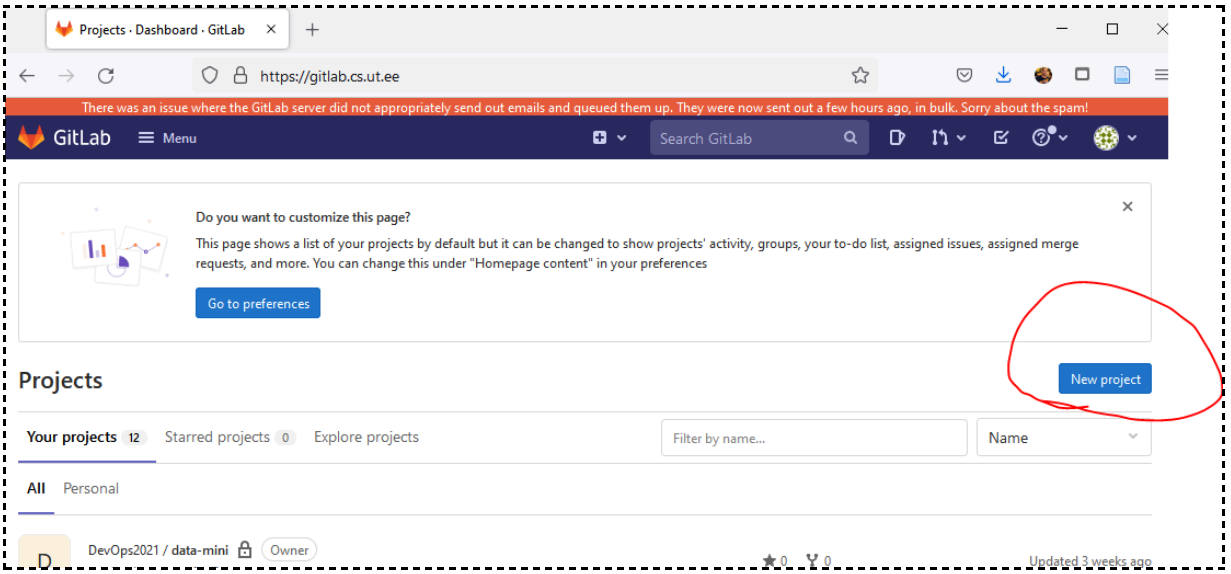
3. To verify, go to your remote gitlab account at https://gitlab.cs.ut.ee/ and see if it is present. The repository should be available at `https://gitlab.cs.ut.ee/dehury/firstrepo`. You should replace `dehury` with your Gitlab username.
4. Screenshot 1.9-2: Take the screenshot of the web UI similar to below:



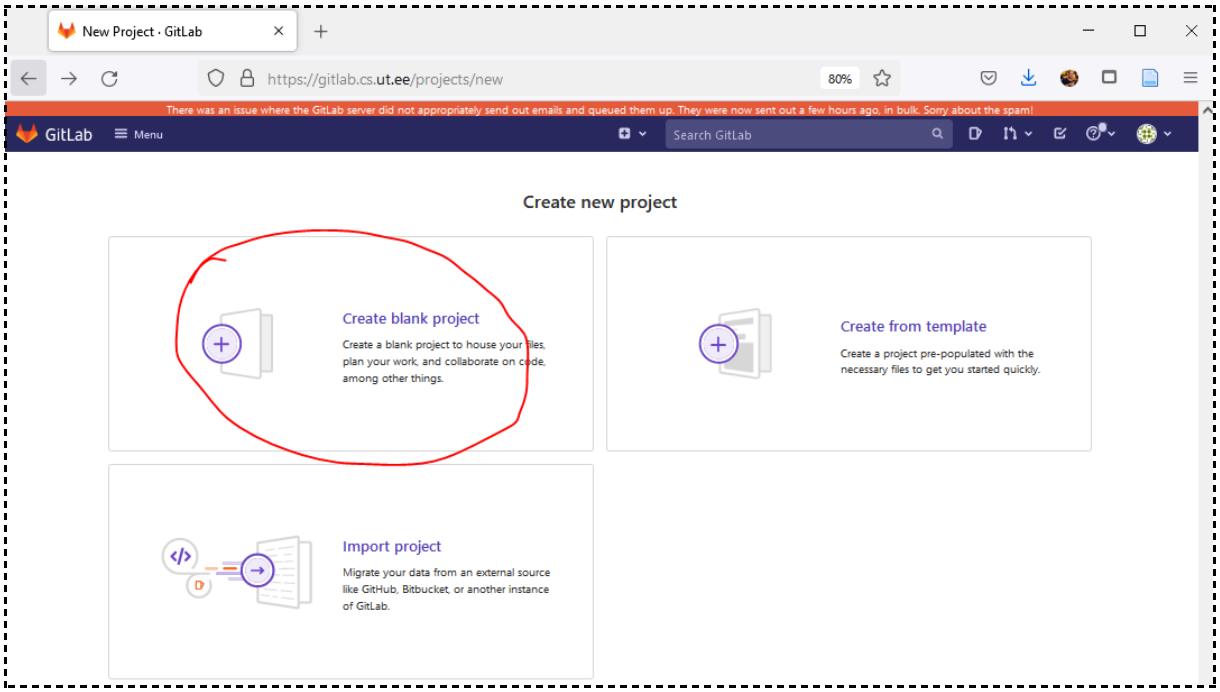

5. Now in the Git Bash terminal come outside the `firstrepo` directory with `cd ..` command

## 1.10: Clone a repo from your remote gitlab account

1. login to your gitlab account https://gitlab.cs.ut.ee/
2. Create a new repo
    - Click on New Project button

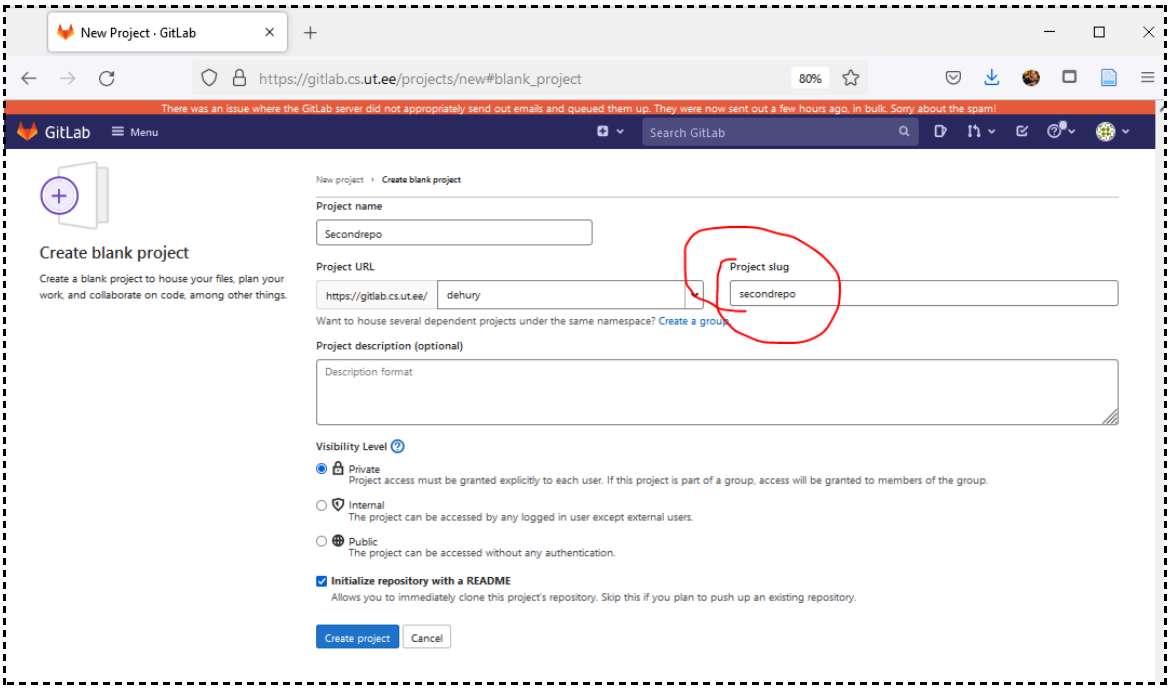



    - Select "`Create blank project`"




    - Enter the project name as `secondrepo` in `Project slug` field and group should be your username(ex: dehury)

- Leave everything else to its default and click on `Create project`.




3. Now go to your git terminal and make sure you are **not** inside `firstrepo` directory.
   - enter clone command: `git clone <url of the second repo>`
     - e.g. git clone https://gitlab.cs.ut.ee/dehury/secondrepo. You should replace `dehury` with your Gitlab username.
     - Screenshot 1.10: Take the screenshot of your terminal output similar to below:

```
$ git clone https://gitlab.cs.ut.ee/dehury/secondrepo
Cloning into 'secondrepo'...
warning: redirecting to https://gitlab.cs.ut.ee/dehury/secondrepo.git/
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.
```

   - At this point, this may ask you for the university's username and password.
   - Now you should see the `secondrepo` directory available in currect directory. Change the current directory to `secondrepo` directory, using `cd secondrepo` command.
   - Here you can see the only default `readme.md` file.

# Exercise 2: Branches and Merging

**Intro**: Branching means you diverge from the main line of development and continue to do work without messing with that main line. The default branch name in Git is master. As you start making commits, you're given a master branch that points to the last commit you made. git init command creates a master branch by default and most people don't bother to change it.

We will use the `firstrepo` repository in this exercise.
Go to the Terminal, `cd firstrepo`. To recap: in the current directory, you have two files: `readme.md` and `LICENSE`.

## 2.1. Creating a New Branch

Create a new branch called `branch-ex2` using `git branch branch-ex2` command. This creates a new pointer to the same commit you're currently on. It is a good practice to use branches rather than the master branch. This allows you to not mess with the main code.

## 2.2. Listing branches

`git branch <options>` command allows you to list, create, or delete branches. List all the branches using the `git branch` command. Here you should see following two branches
`$ git branch`

```
  branch-ex2
* master
```

The * indicates the current branch.

## 2.3. Change the current branch

1. `git checkout <branch name>` is used to switch to another branch. Let's switch to the newly created branch:
    - `git checkout branch-ex2`
2. Now enter git branch command to see the current active branch. The output changes to following:
   `$ git branch`

```
* branch-ex2
  master
```

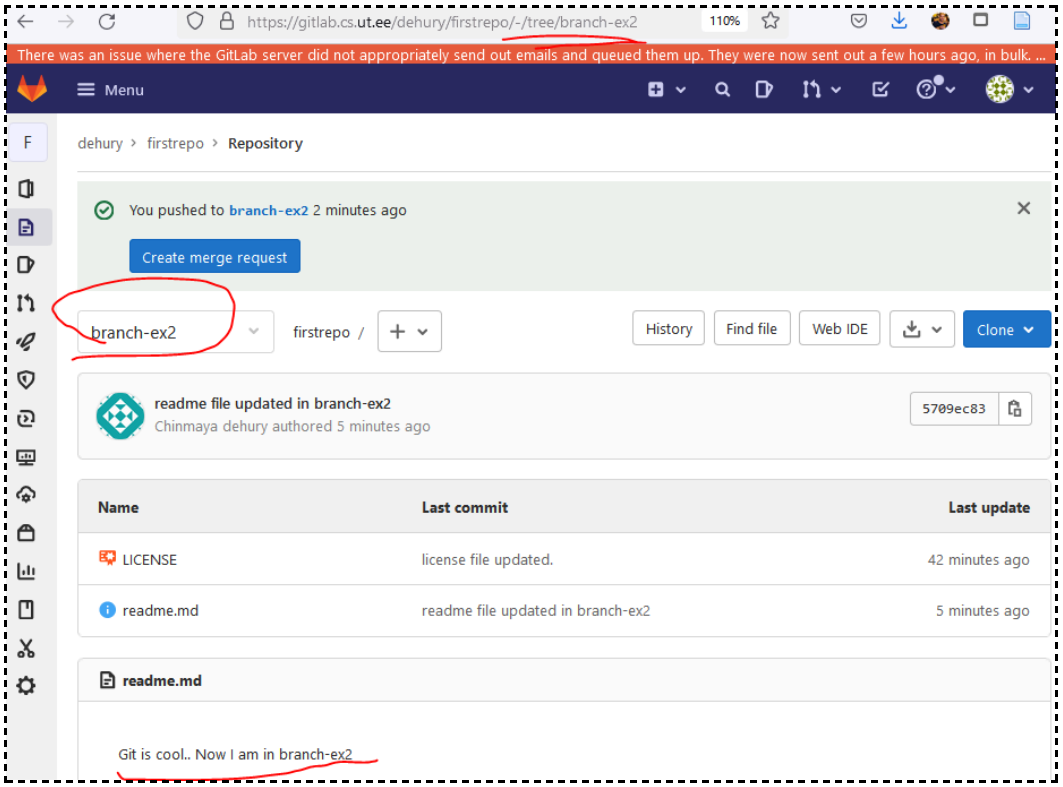
## 2.4. Modify new branch content

1. Now let's append a new line to the `readme.md` file.
    - `echo "Now I am in branch-ex2" >> readme.md`
2. Stage the `readme.md` changes : `git add readme.md`
3. Commit the changes on this branch: `git commit -m "readme file updated in branch-ex2"`
4. Check the repository status using the `git log` command. Here, you should see that the commits are on branch `branch-ex2`.
5. Now check the content of the `readme.md` file. Here you should see the line saying `Now I am in branch-ex2`.
6. Screenshot 2.4-1: Using `git push --set-upstream master` command push all the changes of this branch to your remote gitlab account and take the screenshot terminal output. Sample output given

below:

```
warning: redirecting to https://gitlab.cs.ut.ee/dehury/firstrepo.git/
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 330 bytes | 330.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
remote:
remote: To create a merge request for branch-ex2, visit:
remote:   https://gitlab.cs.ut.ee/dehury/firstrepo/-/merge_requests/new?merge_request%5Bsource_branch%5D=branch-ex2
remote:
To https://gitlab.cs.ut.ee/dehury/firstrepo
 * [new branch]      branch-ex2 -> branch-ex2
```

7. Screenshot 2.4-2: Using your browser, go to your remote Gitlab repo and take the screenshot showing the update status. Sample output given below:

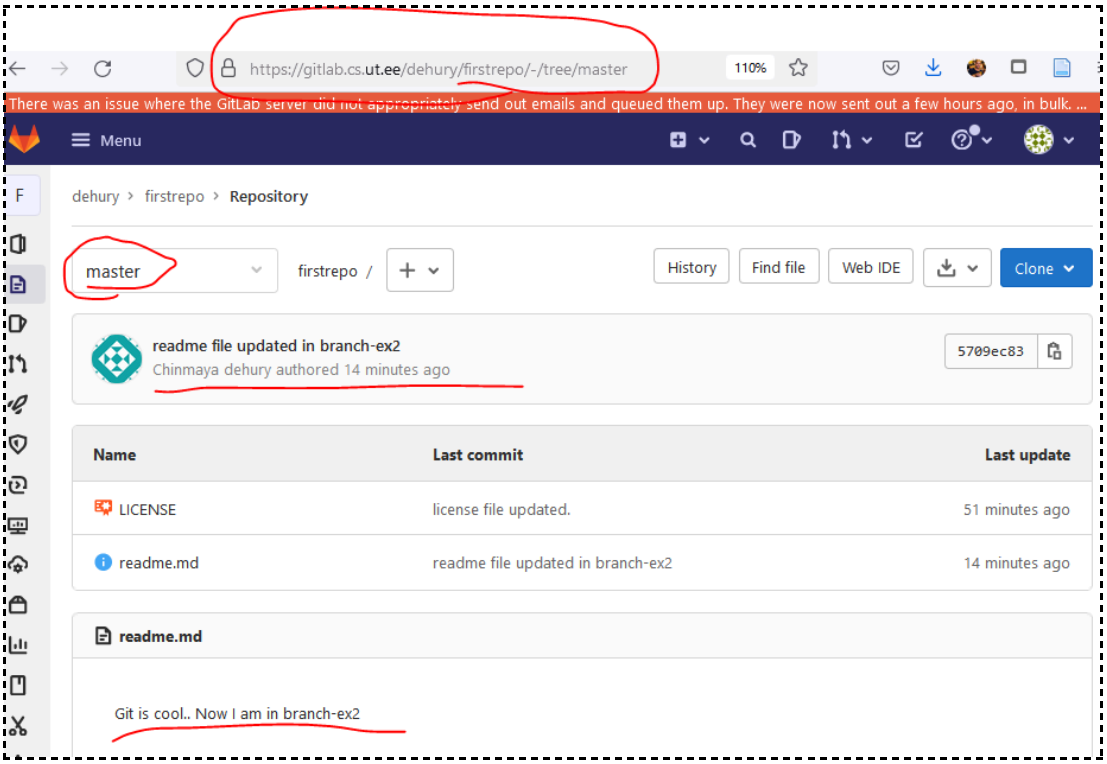



## 2.5. Merge the content of multiple branches

1. Here we will merge the content of the `readme.md` file from `branch-ex2` to master branch using the `git merge <source branch-name>` command.
2. Lets first checkout the master branch: `git checkout master`
3. We are now in the `master` branch. Lets first verify if the line `Now I am in branch-ex2` is present using `cat readme.md` command. As expected, that line should not be present in this branch.
4. Now issue `git merge` command: `git merge branch-ex2 -m "merging readme file to master"`
5. Screenshot 2.5-1: From the terminal push all the changes of this branch to your remote gitlab account and take the screenshot. Sample output given below:

```
warning: redirecting to https://gitlab.cs.ut.ee/dehury/firstrepo.git/
Total 0 (delta 0), reused 0 (delta 0), pack-reused 0
To https://gitlab.cs.ut.ee/dehury/firstrepo
   c1f2e7b..5709ec8  master -> master
```

6. Screenshot 2.5-2: Using your browser, go to your remote Gitlab repo and take the screenshot showing the update status. Sample output given below:




7. Now check the content of the `readme.md` file using `cat readme.md`. The changes from the `branch-ex2` branch should be available in the current `master` branch.

## 2.6. Deleting a branch

1. Make sure that `branch-ex2` and `master` is available using the `git branch` command.
2. Checkout the `master` branch: `git checkout master`
3. Delete the `branch-ex2` branch using following command:
    - `git branch -d branch-ex2`
4. Push all the changes to your remote gitlab account.

5. Question: what will happen if you are checked out at `branch-ex2` while deleting `branch-ex2`?
   You will get the error similar to below:
   `$ git branch -d branch-ex2`

```
error: Cannot delete branch 'branch-ex2' checked out at 'D:/firstrepo'
```

6. Question: what will happen, if the `branch-ex2` is not fully merged with the master branch?
   You will get following error:
   `$ git branch -d master`

```
error: The branch 'master' is not fully merged.
If you are sure you want to delete it, run 'git branch -D master'.
```

# Exercise 3: Handling Merge Conflicts

Conflict arises when you may have made overlapping changes to a file, and Git cannot automatically merge the changes. In this exercise we will see how to handle the conflicts.

### 3.1. Update the readme file with some extra lines.

Lets append some new lines to the `readme.md` file. Make sure that you are inside `firstrepo` in your git terminal.

1. Append two new lines using following commands:
    - `echo "This line is added in master branch." >> readme.md`
    - `echo "This is just an extra line inserted while in master branch." >> readme.md`
2. Stage the file: `git add .`
3. Commit the staged content: `git commit -m "added some extra lines to readme file"`
4. At this point the content of `readme.md` file should be:

```
Git is cool..
Now I am in branch-ex2
This line is added in master branch.
This is just an extra line inserted while in master branch.
```

### 3.2. Create and checkout new branch

1. Lets first create a new branch called `branch-ex3` using the `git branch branch-ex3` command.
2. Switch to or checkout the new branch: `git checkout branch-ex3`
3. Open `readme.md` file and update the third line:
   From: This line is added in master branch.
   To: This line is MODIFIED in BRANCH-EX3 branch.
4. Stage and commit the changes:
    - `git add .`
    - `git commit -m "readme file modified in branch-ex3"`
5. Here, the content of the readme.md file should be:

```
Git is cool..
Now I am in branch-ex2
This line is MODIFIED in BRANCH-EX3 branch.
This is just an extra line inserted while in master branch.
```

### 3.3. Checkout master branch and modify the `readme.md` file

Now lets again modify the same file in `master` branch:

1. First checkout `master` branch: `git checkout master`
2. Verify the content of the `readme.md` file. The content should be as below:

```
Git is cool..
```

```
Now I am in branch-ex2
This line is added in master branch.
This is just an extra line inserted while in master branch.
```

3. Lets update again the third line of `readme.md` file.
   From: This line is added in master branch.
   To: This line is RE-MODIFIED in MASTER branch.
4. Stage and commit the changes
   a. `git add .`
   b. `git commit -m "readme file re-modified in master"`
5. Now the content of `readme.md` file should look like:

```
Git is cool..
Now I am in branch-ex2
This line is RE-MODIFIED in MASTER branch.
This is just an extra line inserted while in master branch.
```

## 3.4. Merge to master branch

Now at this point we will merge the `branch-ex3` branch to `master` branch.

1. Checkout `master` branch: `git checkout master`
2. Merge the `branch-ex3` branch using `git merge branch-ex3` command.
3. Here you will get following similar error:

```
Git is cool..
Auto-merging readme.md
CONFLICT (content): Merge conflict in readme.md
Automatic merge failed; fix conflicts and then commit the result.
```

4. If you now see the content of `readme.md` file, this should look like:
   `$ cat readme.md`

```
Git is cool..
Now I am in branch-ex2
<<<<<<< HEAD
This line is RE-MODIFIED in MASTER branch.
=======
This line is MODIFIED in BRANCH-EX3 branch.
>>>>>>> branch-ex3
This is just an extra line inserted while in master branch.
```

5. The `readme.md` file now contains information to help you find the conflict. The line between `<<<<<<< HEAD` and `=======` represents the line from the `master` branch and the line between `=======` and `>>>>>>> branch-ex3` represents the line from `branch-ex3` branch.
6. Lets remove the line from the `master` branch and keep the line from `branch-ex3` branch.
   For this remove following lines from the `readme.md` file:

```
<<<<<<< HEAD
This line is RE-MODIFIED in MASTER branch.
=======
```

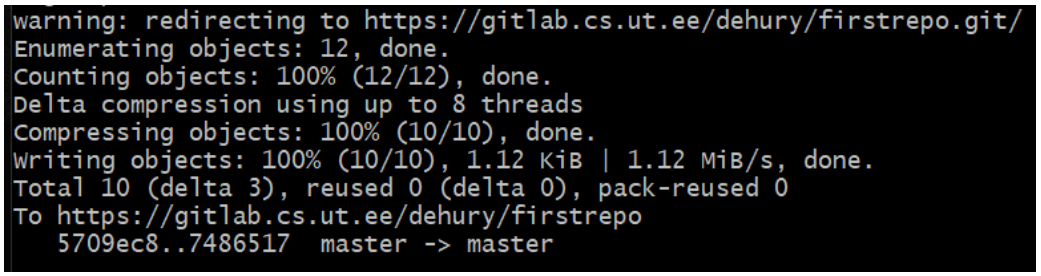
and

```
>>>>>>> branch-ex3
```

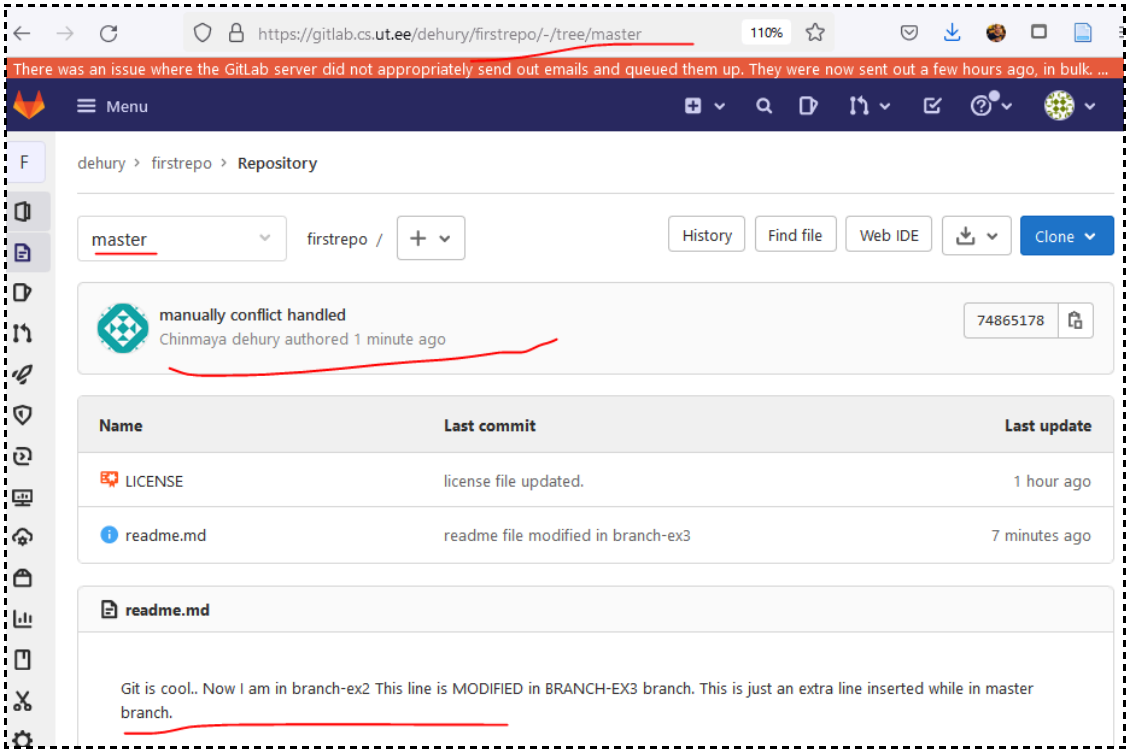
7. Now the `readme.md` file should look like:

```
Git is cool..
Now I am in branch-ex2
This line is MODIFIED in BRANCH-EX3 branch.
This is just an extra line inserted while in master branch.
```

8. Now Stage and commit the changes
   a. `git add .`
   b. `git commit -m "manually conflict handled"`
9. Screenshot 3.4-1: From the terminal push all the changes of this branch to your remote gitlab account and take the screenshot. Sample output given below:



10. Screenshot 3.4-2: Using your browser, go to your remote Gitlab repo and take the screenshot showing the update status. Sample output given below:




# Exercise 4: Forking and merging a branch

**Intro**: A fork is a copy of a repository. Forking a repository allows you to freely experiment with changes without affecting the original project.

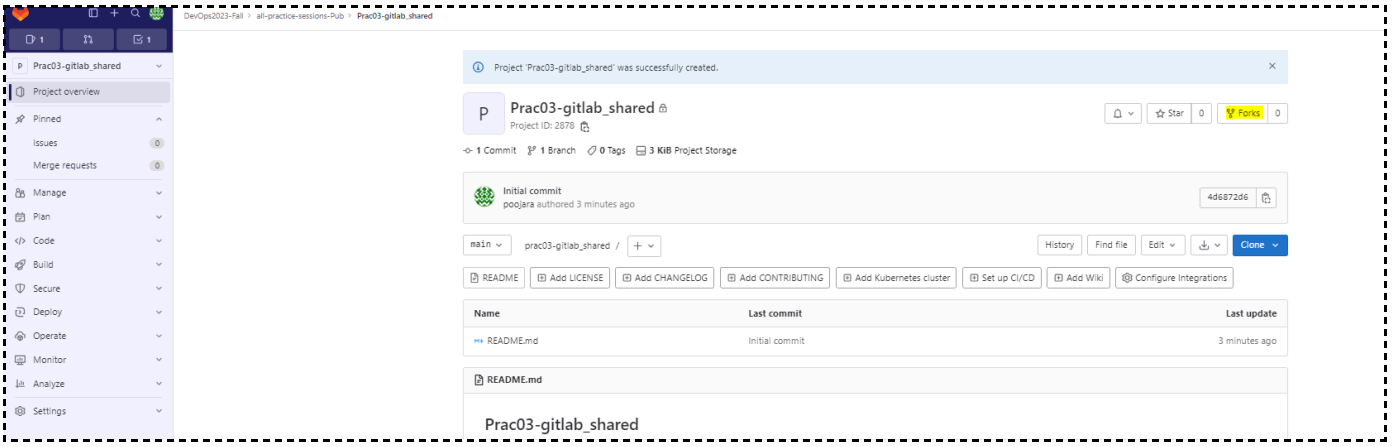
In this exercise your going to complete following tasks:

- Forking the main repository (https://gitlab.cs.ut.ee/devops2023-fall/all-practice-sessions/prac03-gitlab_shared) to your gitlab account.
- You will clone the forked repository and create a branch, modify it by adding your files and merge it with your forked repository.
- Finally, you will send merge request to owner of the main repository (https://gitlab.cs.ut.ee/devops2023-fall/all-practice-sessions/prac03-gitlab_shared )

## 4.1. Forking a project

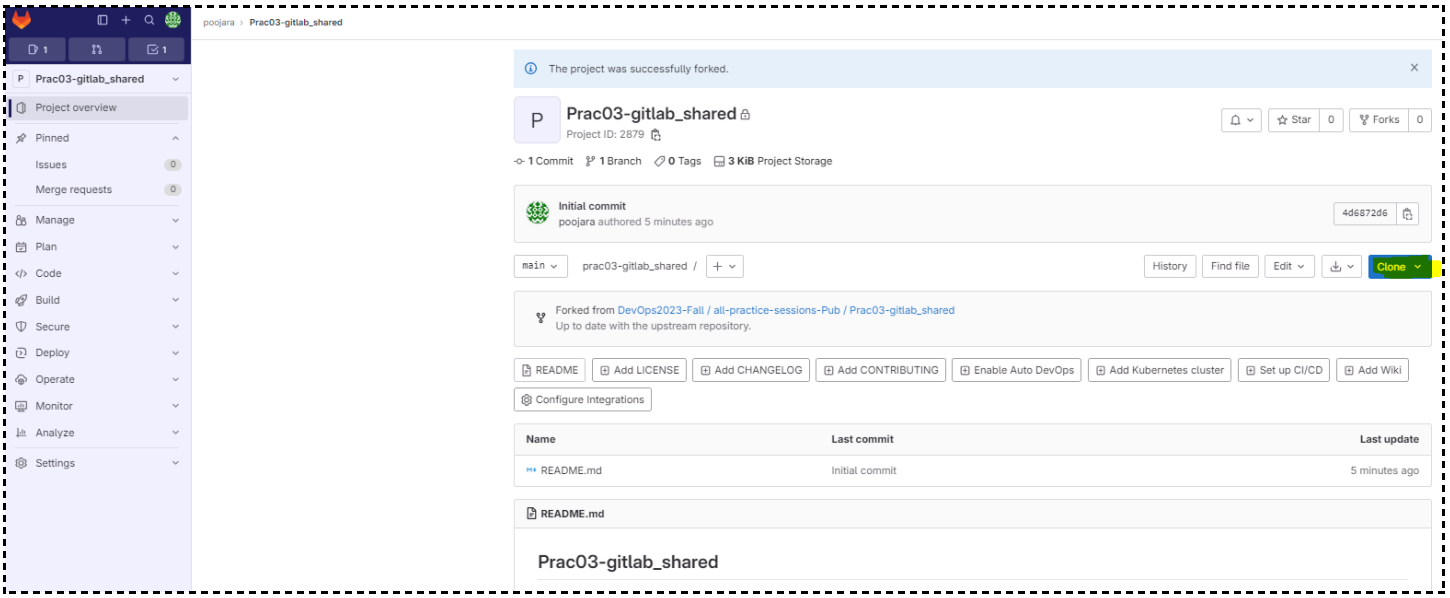
- Login to your remote GitLab account: https://gitlab.cs.ut.ee
- Go to https://gitlab.cs.ut.ee/devops2023-fall/all-practice-sessions/prac03-gitlab_shared

- Fork this repository (GUI)



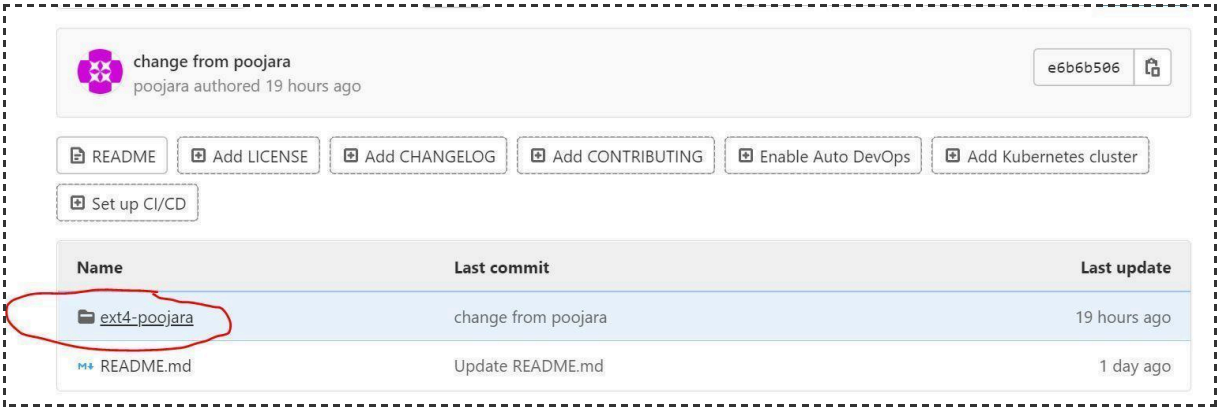

## 4.2. Clone your forked project, create and add your files and merging

- Clone that repository to your local environment using the `git clone <url>` command. The url repository for cloning can be found here




  - Make a new branch `ex4-dehury` using `git branch ex4-dehury`. You should replace `dehury` with your name.
  - Checkout the newly created `ex4-dehury` branch using `git checkout ex4-dehury`. Replace `dehury` with your name.
  - Create a directory: `mkdir dehury && cd dehury`. Replace `dehury` with your name.
  - Create a file `hello.txt` inside the `dehury` directory using the following command.
    `echo "Helloooo, Its <your_name> speaking from DevOps course." > hello.txt`
  - Change to parent directory `cd ..`
  - Stage parent directory `git add .`
  - Commit the changes locally using `git commit -m "change from <your_name>"`
  - See the list of existing remotes `git remote`
  - Add a remote with the name `main` using the command `git remote add main https://gitlab.cs.ut.ee/<your_gitlab_account_username>/prac03-gitlab_shared`
  - Push the changes `git push main`

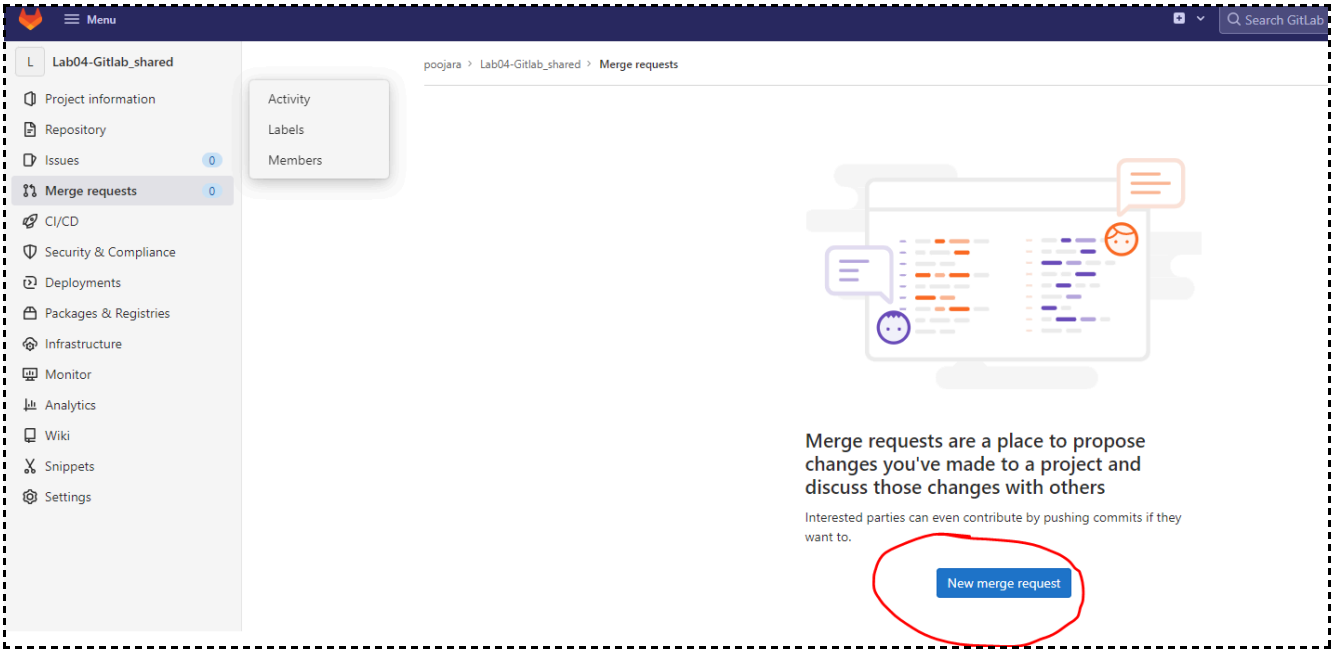
- Check out `git checkout main` and Merge the changes to your repository `git merge ex4-dehury`. Replace `dehury` with your name.
- Push the final changes `git push main`
- After this you should see the following changes in your repo



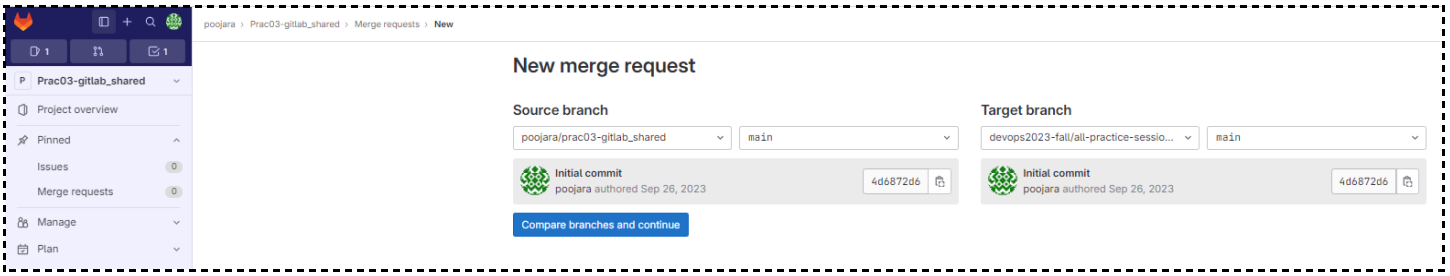

## 4.3. Sending merge request to owner of the main repository

Here, you can send a merge request in two ways: 1)Gitlab GUI 2) git cmd line interface

- Create a merge request using GUI, for this go to your gitlab project and create as shown below:




- Make sure, you added correct projects as shown below




- If everything goes well, PI and TAs will get a merge request notification.

# Deliverable: Uploading Lab01 and Lab02 content

Note!! You can use Visual Studio Code or Git Desktop in your laptop for this task  if needed. Here is the guide for using git commands in Visual Studio Code

In this Exercise you will create projects and upload the content/materials of Lab01 and Lab02. For this we have already created the required group and sent you the invitation email.

- Please check your university email and find the invitation email with the subject "`dehury invited you to join GitLab`".
- The group name and path should be in the following format:
  - `devops2023-fall/students/devops2023Fall-<lastname>-<studyCode>`
  - e.g. `devops2023-fall/students/devops2023Fall-dehury-xxxxxx`
- You are invited to join the group as a "`Maintainer`". You are not authorised to access other students' groups.
- Create the first project with the name "`Prac01 - Etais Intro and Flask web app`"
  - Refer Deliverables section of Lab01.
  - Unzip the zip file that you had uploaded to the course wiki page.
  - Push the "`flask-app`" directory and all the "*screenshots*" to the "`Prac01 - Etais Intro and Flask web app`" project. (PS!! Please ignore python .venv files)
- Create the second project with the name "`Prac02-Docker`"
  - Refer Deliverables section of Lab02.
  - Unzip the zip file that you had uploaded to the course wiki page.
  - Push the "`Flask-webapp`" directory and all the "*screenshots*" to the "`Prac02-Docker`" project.
- Create third project "`Prac03-VCS-Gitlab`"
  - Copy all the screenshots so far you have taken (in the previous exercises) to this directory. This can be done manually using windows explorer or similar environments in other OSs.
  - Add and commit to this project
- [UPDATE] Upload to Course wiki page
  - Upload the screenshot taken wherever mentioned
  - Pack the screenshots (if any) and code (if any) into a single zip file and upload them through the submission form.