TDDC88/TDDC93: Software Engineering Lab 5

Continuous Integration (CI) and Git Branches

Objectives

- To gain fundamental understanding and hands-on experience on CI
- To gain understanding about git branches and manage branches whenever you want to add a new work and push it to the original repository via merge request

This lab will give you some hands-on experience in using continuous integration tools to automate the integration when members of a team push new/modified code into the remote repository. There are several CI tools to choose from, for example Jenkins, GitLab CI, TeamCity, and Travis CI etc...

In this Lab, you will be able to set up a CI system using GitLab, which is one of the more popular CI tools. In GitLab, you can create projects for hosting your codebase, collaborate on code, and automate all sorts of tasks related to building, testing, and delivering or deploying software continuously with built-in GitLab CI/CD. Builds can be triggered by time or event based.

You will be using the FreeCol codebase (FreeCol is a turn-based strategy game), which resides at https://gitlab.liu.se/jesji387/group3, along with GitLab from LiU (https://gitlab.liu.se/), and docker hub images available in https://hub.docker.com/ to build FreeCol application. FreeCol application [**http://www.freecol.org/**] is a Java module, aims to create an open-source version of the game Colonization. FreeCol project's build script, using **Ant,** is configured to build, generate HTML documentation, code coverage reports, automated testing report etc. Apache Ant (https://ant.apache.org/) is a Java library and command-line tool with a number of built-in tasks allowing to compile, assemble, test, and run Java applications

Recommended reading before you start working on this lab:

- Introduction to CI/CD with GitLab
  https://docs.gitlab.com/ee/ci/introduction/index.html#how-gitlab-cicd-works
- GitLab CI/CD Pipeline Configuration Reference
  https://docs.gitlab.com/ee/ci/yaml/README.html
- Creating and Tweaking GitLab CI/CD for GitLab Pages-
  https://docs.gitlab.com/ee/user/project/pages/getting_started_part_four.html
- JUnit test reports - https://docs.gitlab.com/ee/ci/junit_test_reports.html

Table of Contents

Part A – Tutorial on how to set up a CI using Gitlab 3

1. Create a “helloworld” project in GitLab 3

2. Clone your remote repository 5

Questions: 6

3. Working on your local repository 7

4. Gitlab CI configuration for “helloworld” project 8

Part B – GitLab CI Configuration and Git Branches 11

1. Setting up a shared Git repository 11

2. Forking the FreeCol repository 13

3. Cloning the repository 14

4. Run build scripts in FreeCol in your local computer 16

5. Set up Continuous Integration in GitLab 17

6. Step 4. Create a merge request in GitLab to the original repository 20

7. Merge the proposed changes 21

8. Task 5. Fixing the test cases 21

9. Merge test fix into the master branch 22

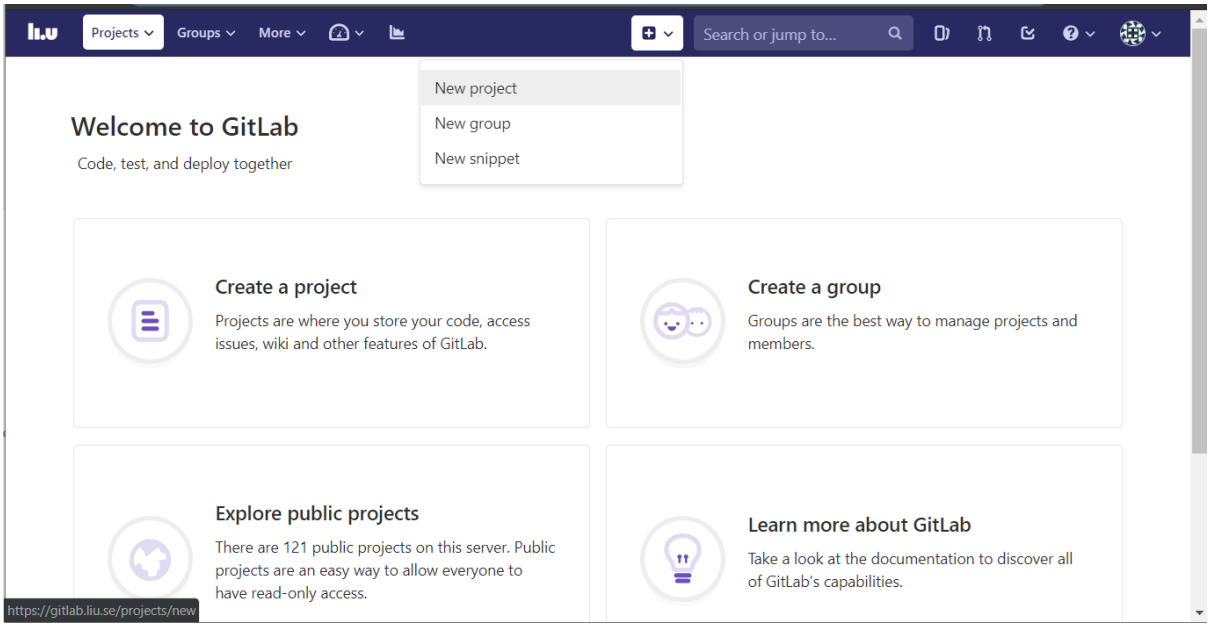
Examination 24

Part A – Tutorial on how to set up a CI using

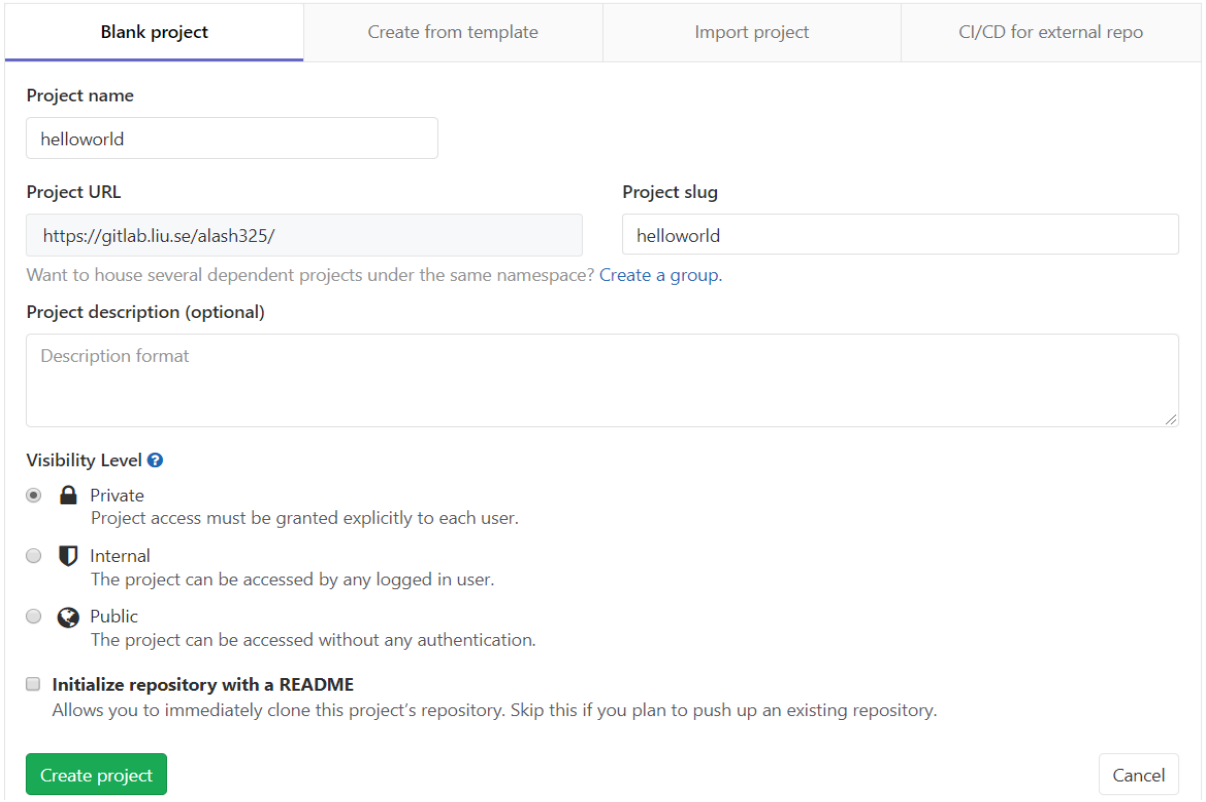
Gitlab

1. Create a “helloworld” project in GitLab

1. Sign in on **https://gitlab.liu.se/** using your LiU account.
2. In your dashboard, click the green **New project** button or use the plus icon in the navigation bar.

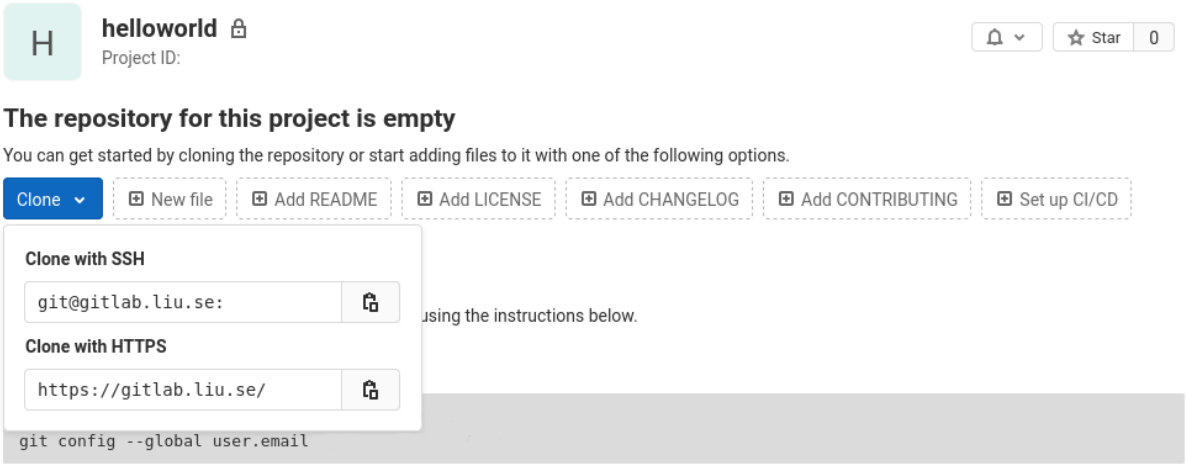


This opens the **New project** page.



3. On the **New project** page, choose the **blank project** tab.
4. On the **Blank project** tab, provide the following information:
    - The name of your project (i.e., **helloworld**) in the **Project name** field.
    - Select the Initialize repository with a README option to create a README file.
5. Click **Create project.** Now your Git repository is initialized with a default branch (**master**).

2. Clone your remote repository



To start working locally on your remote repository, you must clone (download) a copy of its files to your local computer. You can clone it via HTTPS:

- From your **helloworld** project click on the **Clone** button
- Copy the URL from "**Clone with HTTPS"** field
- Open a terminal in the directory you wish to clone the repository files into, and run the following command:

  git clone [repository_URL]

  where ***[repository_URL]*** is the URL (remote repository path) you copied.
- Enter your credentials if asked.

  Once completed, the following actions occur:
- A new folder called **hellworld** initialized as a Git repository is created in your local computer
- A remote named ***origin*** is created, pointing to the URL you cloned from. Go to **helloworld** folder and type ***git remote -v*** and press **Enter**. You will see the current configured remote repository:

  origin https://gitlab.liu.se/<USERNAME>/helloworld.git (fetch)

  origin https://gitlab.liu.se/<USERNAME>/helloworld.git (push)
- All of the repository's files and commits are downloaded there (for now only the README file)

- The default branch (usually called master) is checked out. Type ***git branch*** and press **Enter**. You will see your branches. A * will appear next to the currently active branch: *** master**.

Questions:

Write few lines (i.e., what, and why mostly) about each of the following terms, with respect to GIT:

1) Origin
2) Remote
3) Branching (master or other) – Creating a new branch
4) Forking
5) Merge Request
6) Continuous Integration
7) Continuous Delivery
8) Making a pull request
9) Build Script
10) .yml file in CI workflow

3. Working on your local repository

1. Create a file called **HelloWorld.java** in you project root, which contains the simplest program of Java printing "Hello World" to the screen.

```
class HelloWorld
{
    public static void main(String args[])
    {
        System.out.println("Hello, World");
    }
}
```

2. Push all your changes to Gitlab (your remote repository):
   - Run the command "git status" to know how many files have been added/changed.
   - Add the file to your local repository and stage it for commit to your local repository:

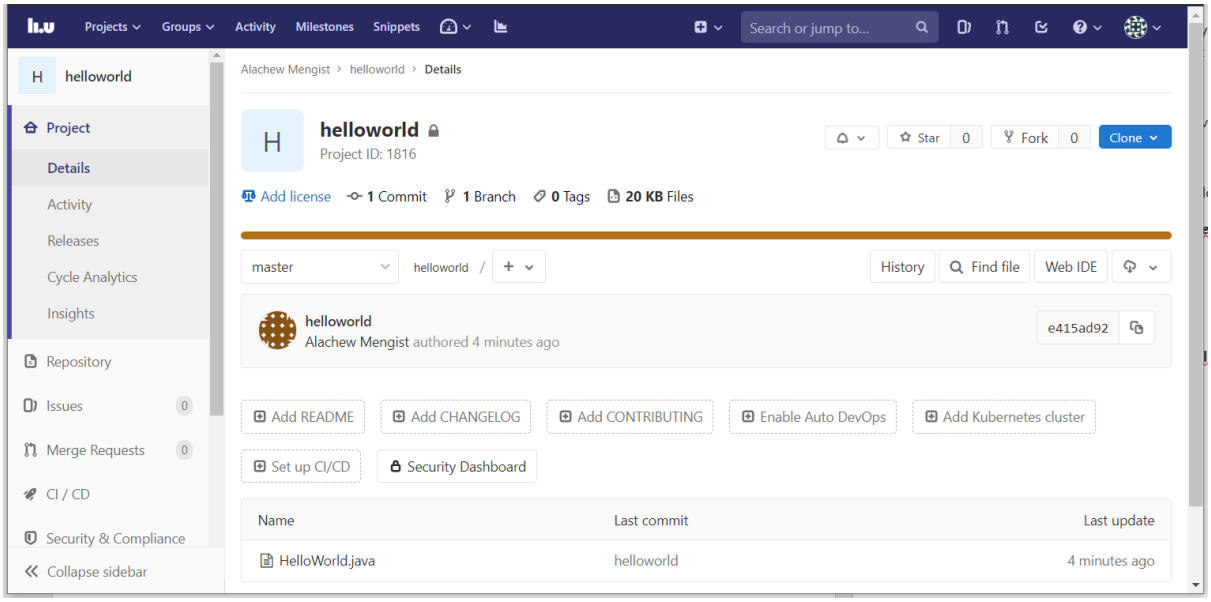
     git add . or **git add HelloWorld.java**

     Note: The . character means all file changes in the current directory and subdirectories.
   - Commit the file that you've staged in your local repository:

     git commit -m "helloworld", where **-m** stands for message, you want to associate with these changes so you remember, why you made these changes.
   - Push all commits in your local repository to GitLab (remote repository)

     git push ***[name_of_your_remote] [name_of_your_branch]***

On successful completion, your new file “HelloWorld.java” will be added to your remote repository in GitLab webpage as well as on local computer.



4. Gitlab CI configuration for “helloworld” project

- Create **.gitlab-ci.yml** and save it in your root directory of **“helloworld”** project. The

```
image: openjdk:8
stages:
  - build
build:
  stage: build
  script:
   - javac HelloWorld.java
   - java  HelloWorld
  artifacts:
   paths:
   - ./HelloWorld.*
```

contents of .yml file is as follows:

NOTE: YAML does **not** allow tab indentation, so use 2 spaces instead. If you are using vim to create the ci file, run **set ts=2 sw=2 et** and add a line containing **# vi: ts=2 sw=2 et** at the bottom of the file. Ensure that the file **~/.vimrc** contains the line **set modeline**.

Question: Explain each line in the figure above.

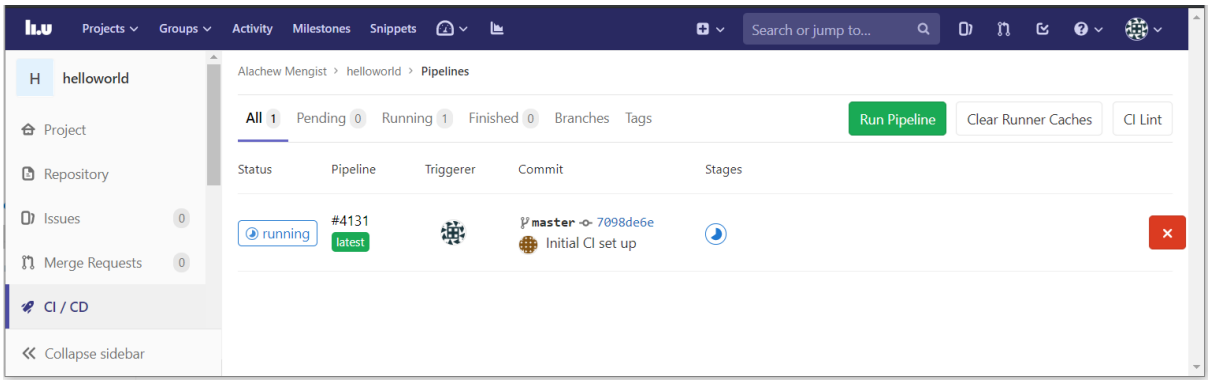
- Push all your changes to Gitlab (your remote repository)

  git add .

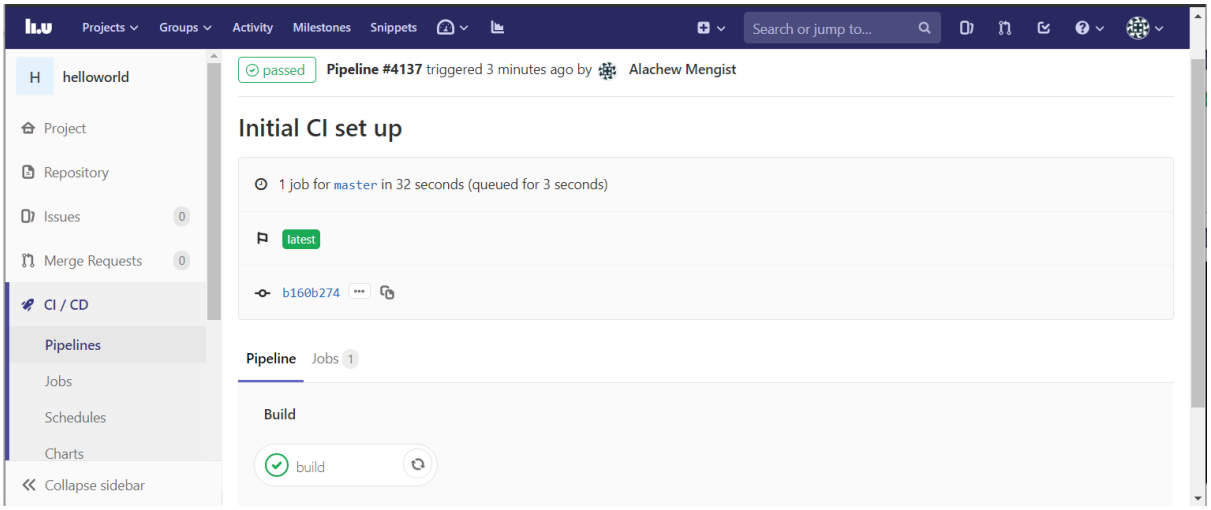
  git commit -m "initial CI"

  *git push **[name_of_your_remote] [name_of_your _branch]***

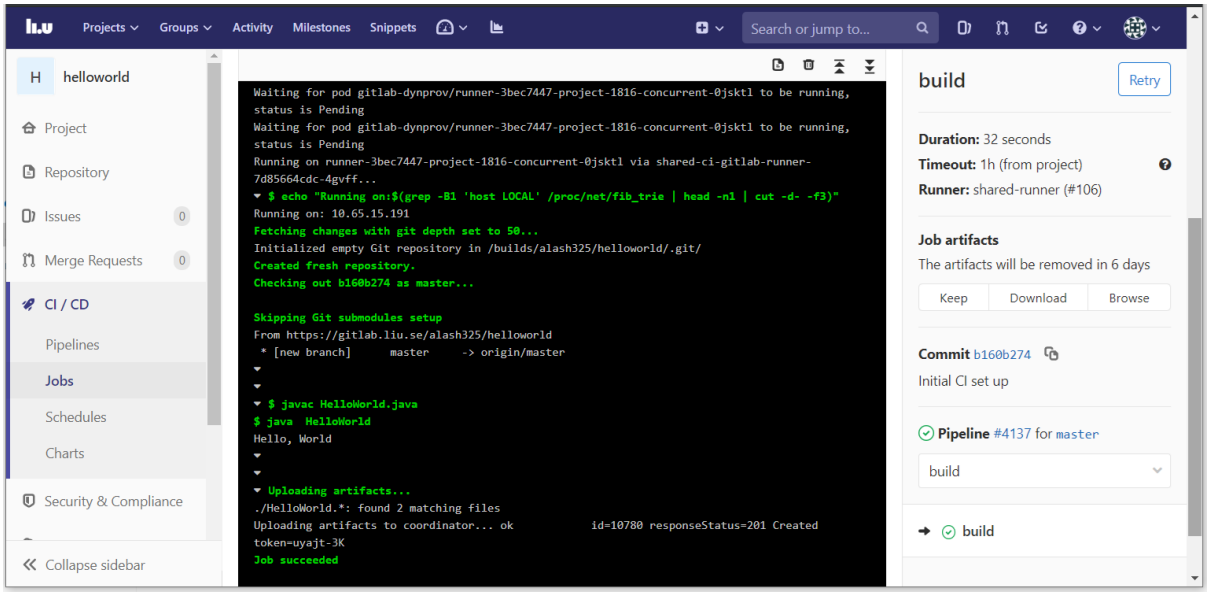
On successful completion, GitLab CI automatically starts to execute the jobs you've set. Go to option on left side on gitlab.liu.se "CI/CD" and click on "pipelines". You should see the status of you last commit changes from **pending** to either **running**, **passed** or **failed** as shown below (see your Gitlab repository).



You can view all pipelines by going to the **Pipelines** page in your project.



In GitLab CI, Runners run the code defined in **.gitlab-ci.yml.** GitLab CI not only executes the jobs you've set, but also shows you what's happening during execution, as you would see in your terminal. You can also view if artifacts were stored correctly using the build artifacts browser that is available from the build sidebar.


Projects
Groups
Activity
Milestones
Snippets
Search or jump to...
H
helloworld
Project
Repository
Issues
0
Merge Requests
0
CI / CD
Pipelines
Jobs
Schedules
Charts
Security & Compliance
Collapse sidebar
Waiting for pod gitlab-dynprov/runner-3bec7447-project-1816-concurrent-0jsktl to be running, status is Pending
Waiting for pod gitlab-dynprov/runner-3bec7447-project-1816-concurrent-0jsktl to be running, status is Pending
Running on runner-3bec7447-project-1816-concurrent-0jsktl via shared-ci-gitlab-runner-7d85664cdc-4gvff...
$ echo "Running on:$(grep -B1 'host LOCAL' /proc/net/fib_trie | head -n1 | cut -d- -f3)"
Running on: 10.65.15.191
Fetching changes with git depth set to 50...
Initialized empty Git repository in /builds/alash325/helloworld/.git/
Created fresh repository.
Checking out b160b274 as master...
Skipping Git submodules setup
From https://gitlab.liu.se/alash325/helloworld
* [new branch] master -> origin/master
$ javac HelloWorld.java
$ java HelloWorld
Hello, World
Uploading artifacts...
./HelloWorld.*: found 2 matching files
Uploading artifacts to coordinator... ok id=10780 responseStatus=201 Created
token=uyajt-3K
Job succeeded
build
Retry
Duration: 32 seconds
Timeout: 1h (from project)
Runner: shared-runner (#106)
Job artifacts
The artifacts will be removed in 6 days
Keep
Download
Browse
Commit b160b274
Initial CI set up
Pipeline #4137 for master
build
build

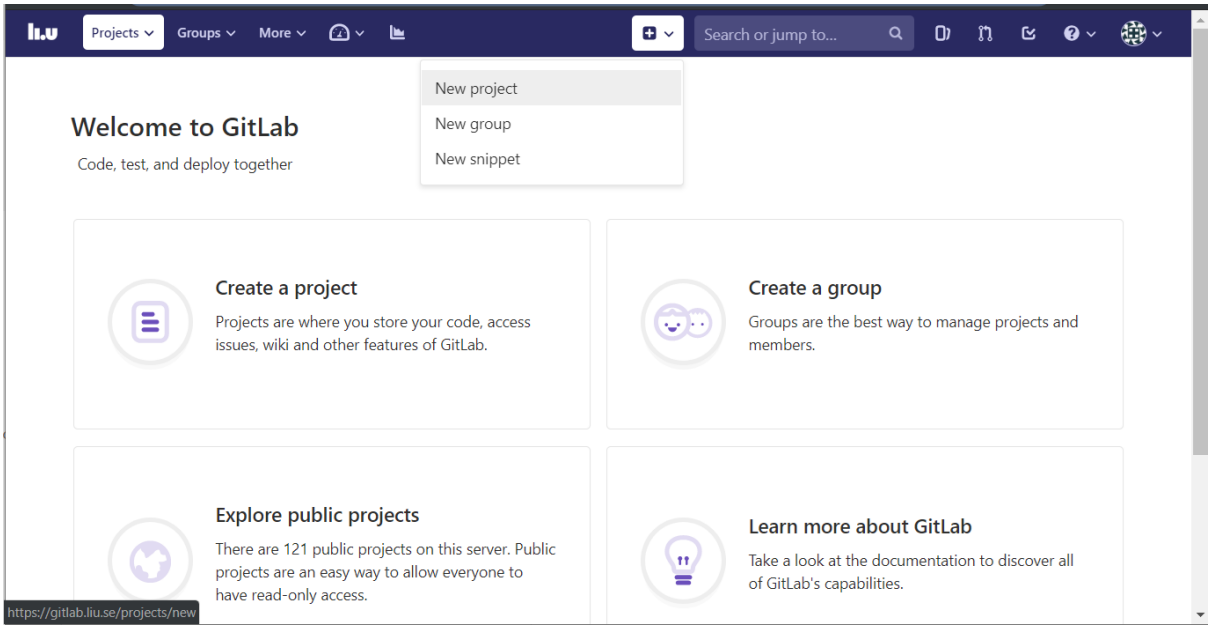
Part B – GitLab CI Configuration and Git Branches

In Part B, you have to work in pair and the project owner (In this case student-A) want the GitLab CI tool to continuously integrate, build and test his FreeCol project automatically triggered by push event, and publish the test results.

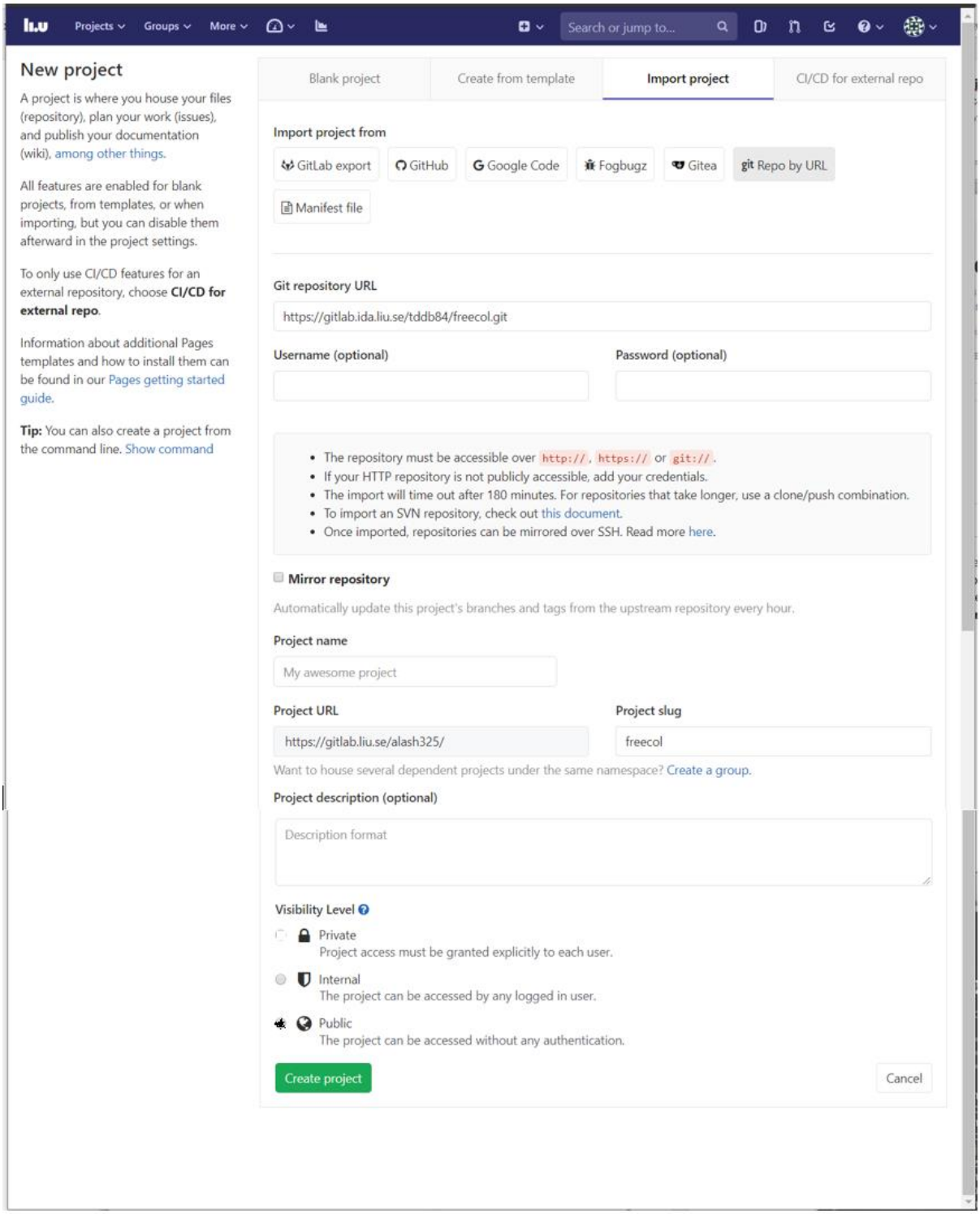
1. Setting up a shared Git repository

In this task, you need to import an existing FreeCol repository via HTTP by providing the Git URL from the New Project page:

- Sign in on https://gitlab.liu.se/ using "Student_A" LiU account



- From your GitLab dashboard click **New project**
- Switch to the **Import project** tab
- Click on the **Repo by URL** button
- Fill https://gitlab.liu.se/jesji387/group3.git in the "Git repository URL". Use Public Checkbox. By Default, it is private.
- Click **Create project** to begin the import process



Once completed, you will be redirected to your newly created FreeCol project accessible at https://gitlab.liu.se/**<student_A_LiU_ID>/**freecol, which will be your original remote repository.

Your task is now to configure CI for FreeCol project using built-in GitLab CI/CD and propose changes to the original repository **(i.e., to the GitLab repository reside in student A)**. You are expected to set up the following stages: build, test, and publish HTML test report page in GitLab. The second requirement for this lab is to create a new branch with git whenever you want to add a new work and push it to the remote/original repository via create merge request.

2. Forking the FreeCol repository

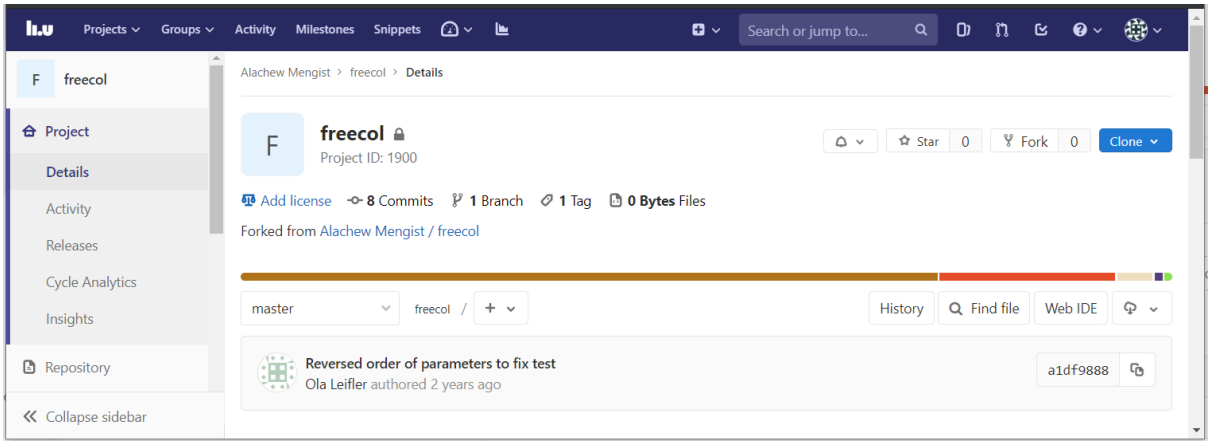
A fork is a copy of original repository. Forking a repository allows you to freely experiment with changes without affecting the original project. In order to push to the original repository, you will push to your own forked repository and create a merge request from your forked repository in GitLab.

- Sign in on https://gitlab.liu.se/ using "Student_B" LiU account
- Navigate to the FreeCol repository of student_A in your browser (https://gitlab.liu.se/<student_A_LiU_ID>/freecol)
- In the top-right corner of the navigated page, click **Fork**
- Select your namespace to fork the project

  Now if you look at the URL of your forked repository it is changed into <student_B_LiU_ID>:

https://gitlab.liu.se/<student_B_LiU_ID>/freecol

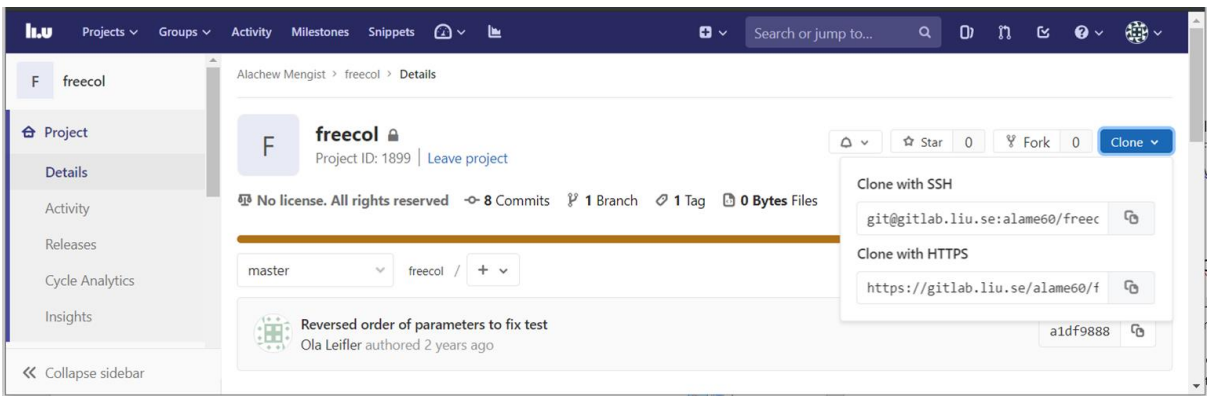
It also shows you where your repository is forked from (see Figure below).



3. Cloning the repository

Log in on the lab terminal as **Student_B**.

To start working locally, you must clone (download) a copy of the original repository files (available in Student_A GitLab repository) to <Student_B> local computer. You can clone it via HTTPS:

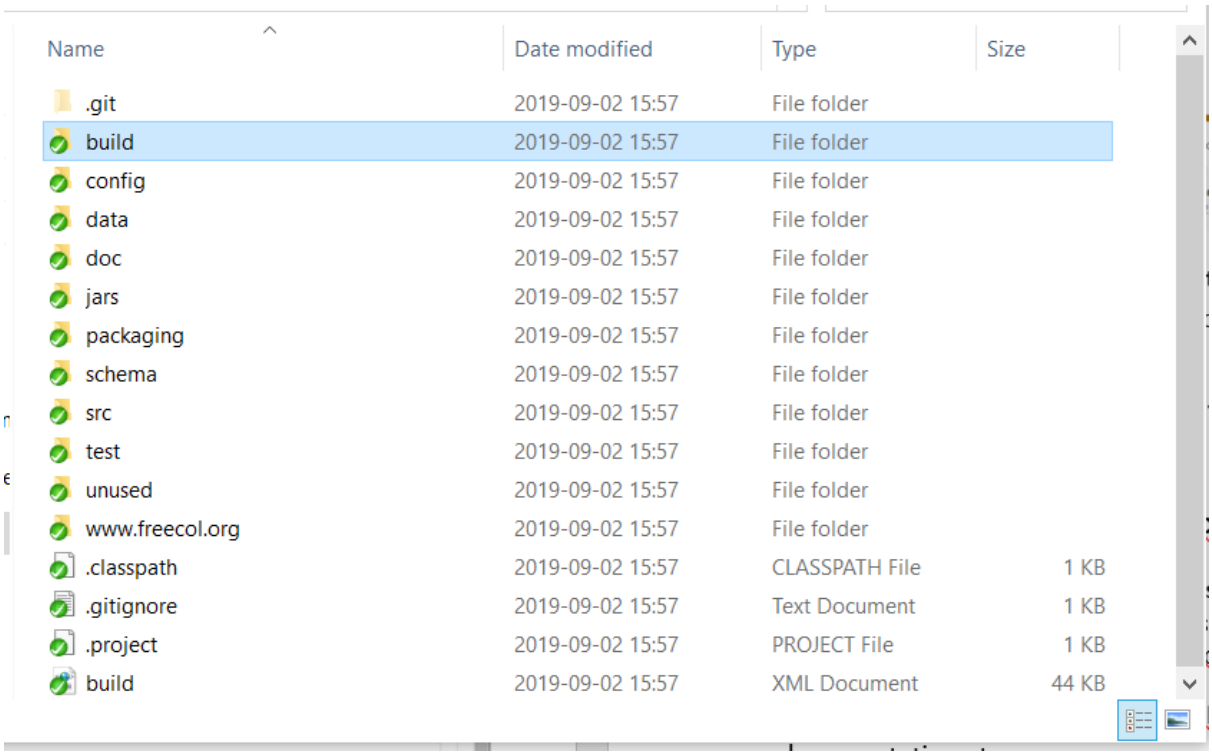


- Go to https://gitlab.liu.se/<student_A_LiU_ID>/freecol

- Click on the **Clone** button
- Copy the URL from "**Clone with HTTPS"** field
- Open a terminal in the directory you wish to clone the repository files into and run the following command.

  git clone [repository_URL] , where ***[repository_URL]*** is the URL (original remote repository path) you copied.

After cloning the repository successfully, you should see the copy of files in your local git repository as shown below (see the directory where you cloned the repository).



Now add the forked repository as well:

- Run **git remote add <Student_B> <URL to Student_B's fork>**

git remote -v shoud now output the following lines:

origin https://gitlab.liu.se/<Student_A>/freecol.git (fetch)

origin https://gitlab.liu.se/<Student_A>/freecol.git (push)

<Student_B> https://gitlab.liu.se/<Student_B>/freecol.git (fetch)

<Student_B> https://gitlab.liu.se/<Student_B>/freecol.git (push)

4. Run build scripts in FreeCol in your local computer

As mentioned in the introduction, FreeCol application uses the **Apache Ant** build script. By default, **Ant** uses **build.xml** as the name for a buildfile. You can find the FreeCol buildfile in your local repository (see ./build.xml). It includes targets for building FreeCol, distribution packages, running tests, creating documentation etc.

Now try to build, run tests, and create documentation for FreeCol java application using the **Ant** build script:

- Open a terminal in the directory where your FreeCol project is copied and run the following command:
- **ant build** to build the FreeCol app
- **ant -lib test/lib/junit.jar -Dtest=AllTests testall** to run the JUnit tests and create a browsable HTML report (see the result **index.html** file under **./build/report/**).

  NOTE: if the "ant" command does not work. Skip to next stage. We have provided the docker image including "ant" below.

5. Set up Continuous Integration in GitLab

The workflows required to have working CI and pass this lab is summarized as follows:

1. Create a new branch
2. Add your code for CI configuration. You CI configuration should include the following:
   - Build the FeeCol application.
   - Run the FreeCol test suite.
   - Store the result as a build artifact (for the test suite).
   - Publish the test results with GitLab pages.
3. Push your changes to your forked repository.
4. **Create *merge request*** in GitLab to the original repository (i.e., to student A).
5. Merge the proposed changes in GitLab (from student_A Gitlab repository).

You are free to work with your own starting point but If you have trouble finding a starting point, follow the following steps:

1. Create a new branch called **"CI_<student_B_id>"** on your local machine and switch in this branch:

   git checkout –b *<your_new_branch_name>* where ***<your_new_branch_name>*** is **"CI_<student_B_id>"**

   This will switch your local git repository to a new branch called **"CI_<student_B_id>".** You can verify this by running **git branch** in your terminal. The output shall be:

   * CI _<student_id>

     Master

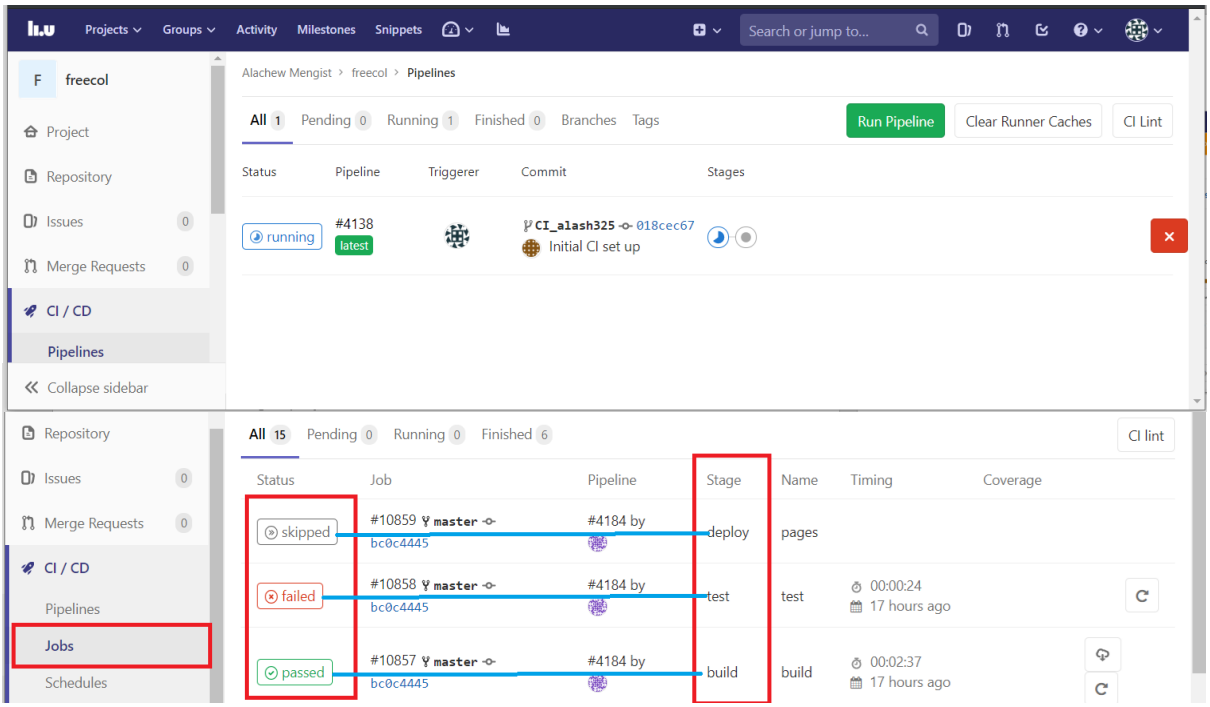
2. Create a new file called **.gitlab-ci.yml**. The first line of yaml file i.e a Docker image to run your script written in .gitlab-ci.yml file is given below.

   image: alash325/javaant:latest

3. Complete other parts using tutorials given in Part A and reference materials or your own material.

4. Once you are satisfied with you CI configuration, push your changes to your forked repository with **git push <Student_B> CI_<student_b_id>**.

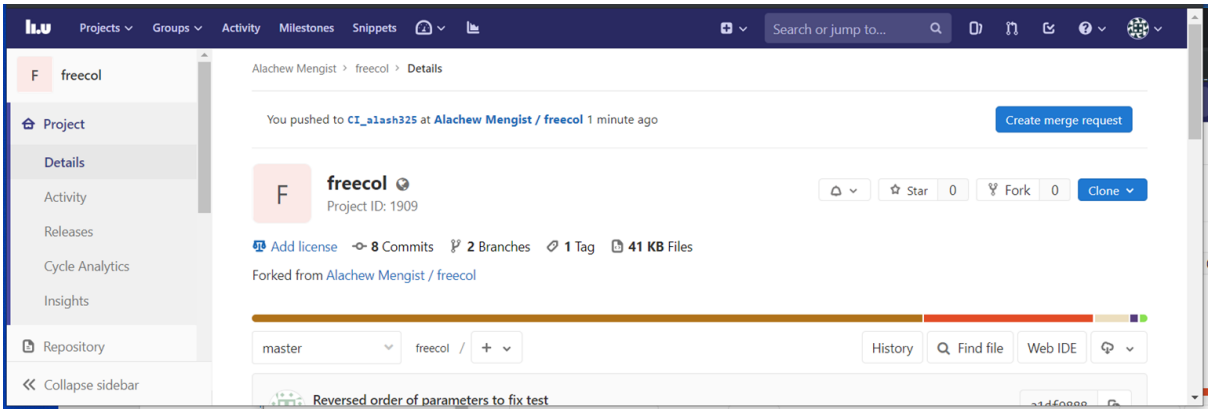
On successful completion, GitLab CI automatically starts to execute the jobs you've set as shown below (Go to https://gitlab.liu.se/<student_B_LiU_ID>/freecol and see your Gitlab repository).



Wait until all the jobs (stages) are executed. Once completed, go to **CI/CD->Jobs** from your project page and see the result for all the jobs you were expected to complete (i.e., build, test, publish html pages). If the results are as shown below in the graph which can be interpreted as: The CI for build job is **passed**, test job is **failed**, and publish html page **skipped**, then go to **Step 4** otherwise try to fix your CI configuration until you reached the expected results. The reason for the test job failure is due to the sound mixer is not available in our computer system. However, for the GitLab runner to execute and publish html page for the test, you are expected to fix those tests otherwise GitLab runner skips to execute the deploy (publish html page) stage. You are going to do this in **step 8**.

6. Step 4. Create a merge request in GitLab to the original repository

Go to https://gitlab.liu.se/<student_B_LiU_ID>/freecol



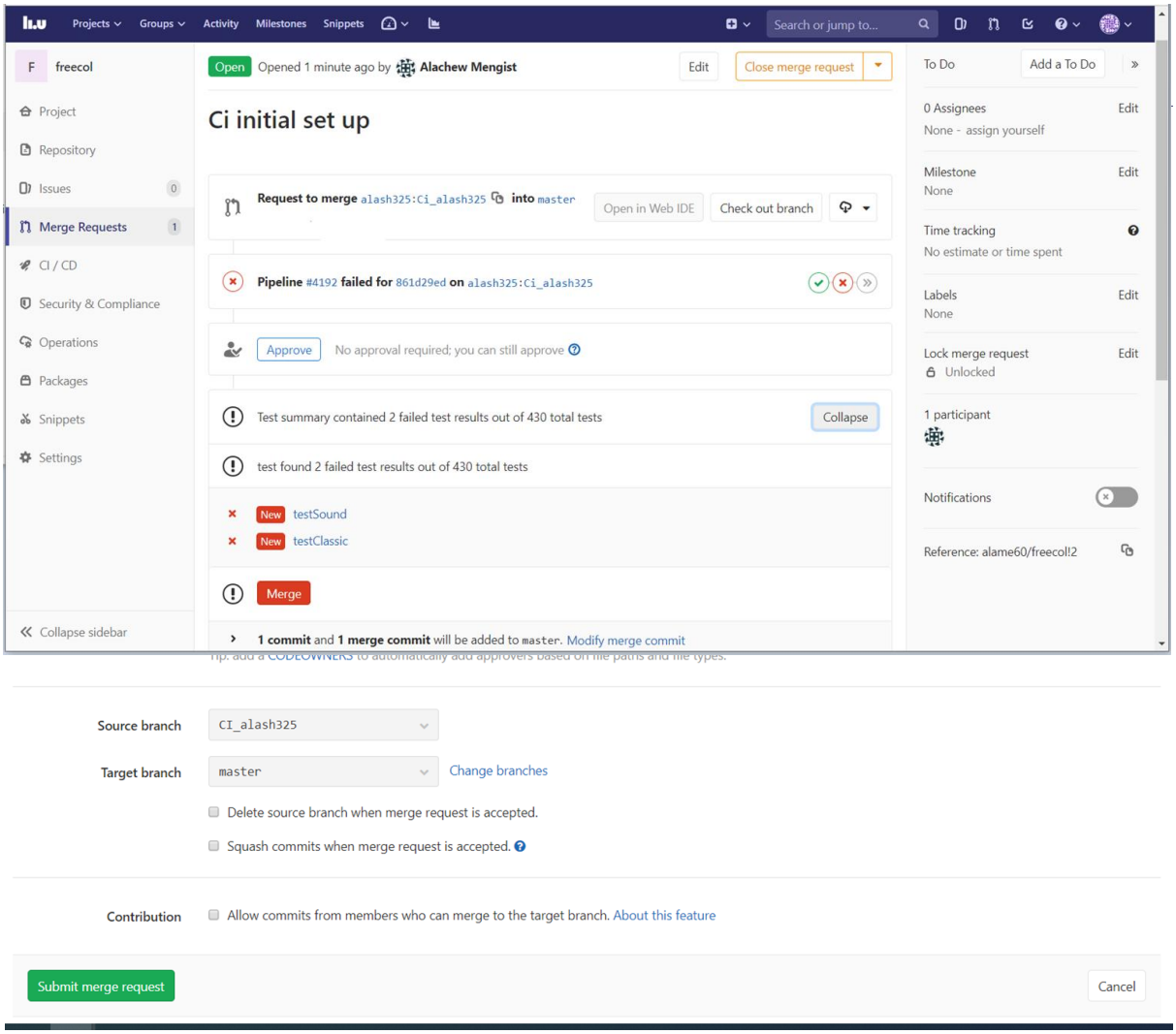
- Click on **Create merge request** button
- Click on **Submit merge request** button

7. Merge the proposed changes

- Go to your original repository

https://gitlab.liu.se/<student_A_LiU_ID>/freecol

- Go to the **Merge Requests** tab and open the merge request. Once you open the merge request you will be redirected to the page like the figure below.



- Click on Merge to merge the proposed changes to your repository

8. Fixing the test cases

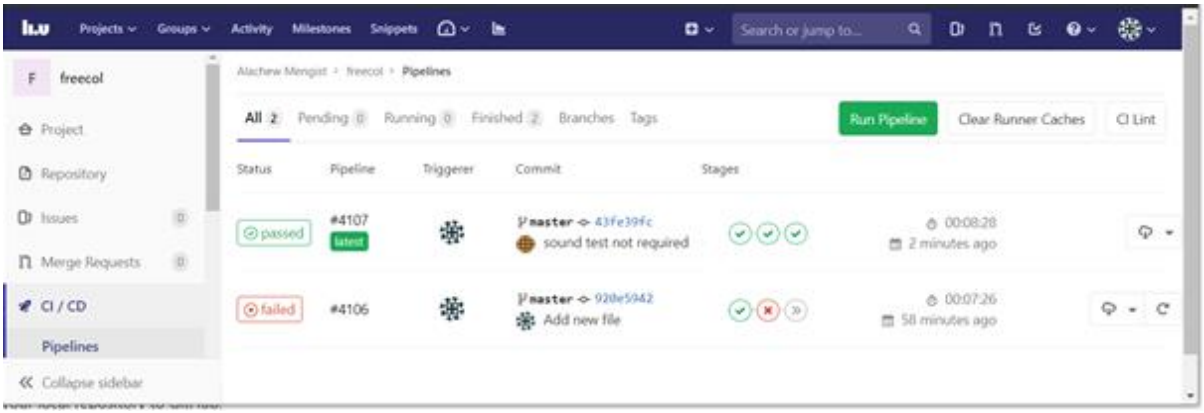
As mentioned, the reason for the test job failure is due to the sound mixer is not available in our computer system. However, for the GitLab runner to execute and publish html page for the test, you are expected to fix those tests. In this lab, we will fix just by removing those tests. Instructions are given below.

1. Check out to the original repository i.e., git checkout master
2. Pulling changes from a remote repository to retrieve new work done by other people and combines your local changes with changes made by others

3. Create a new branch called "Fix_tests" and checkout your new branch
4. Go to **AllTests.java** in ./test/src/net/sf/freecol/common/AllTests.java and comment the following line of code:

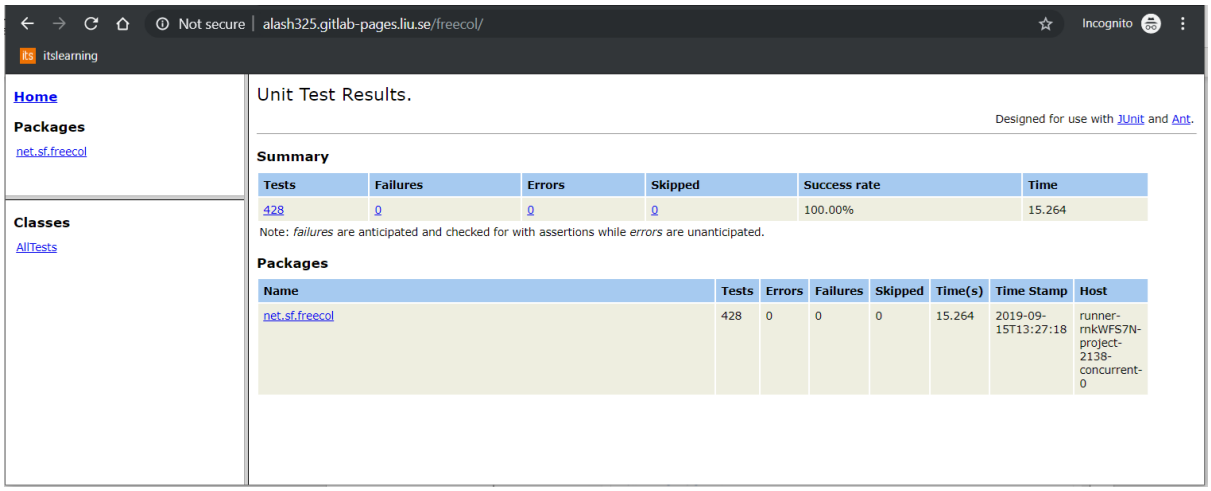
   // suite.addTest(net.sf.freecol.common.sound.AllTests.suite());
5. Push your changes (See **Step 3 in Task 4**)

After pushing your changes successfully, GitLab CI automatically starts to execute the jobs you've set and you can view the status of the Pipelines.



After successful Pipelines you can view your clickable HTML pages by going to your projects **Settings->Pages**.

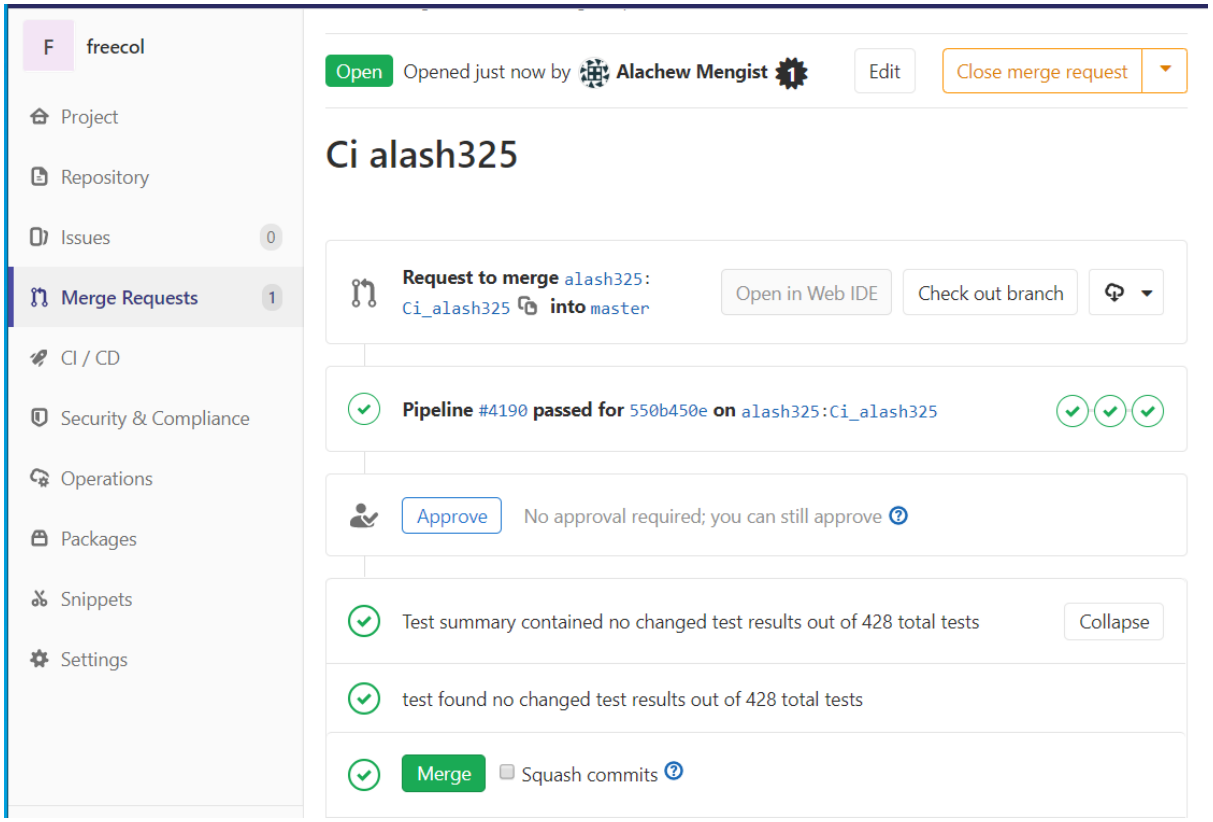
You can view your HTML report by clicking the pages your found.



9. Merge test fix into the master branch

1. Create a merge request (See **Step 5 in Task 4**)

2. Merge the requested changes from your original repository (See **Step 5 Task 4**)



Examination

When you are done with all tasks and ready to demonstrate your solutions, contact your assistant during a lab occasion. Be prepared to show the workflows described in Task 4 and answers to questions. The lab assistants may ask you to send in these answers for later evaluations.