CPSC 2720 – Assignment 0

Overview

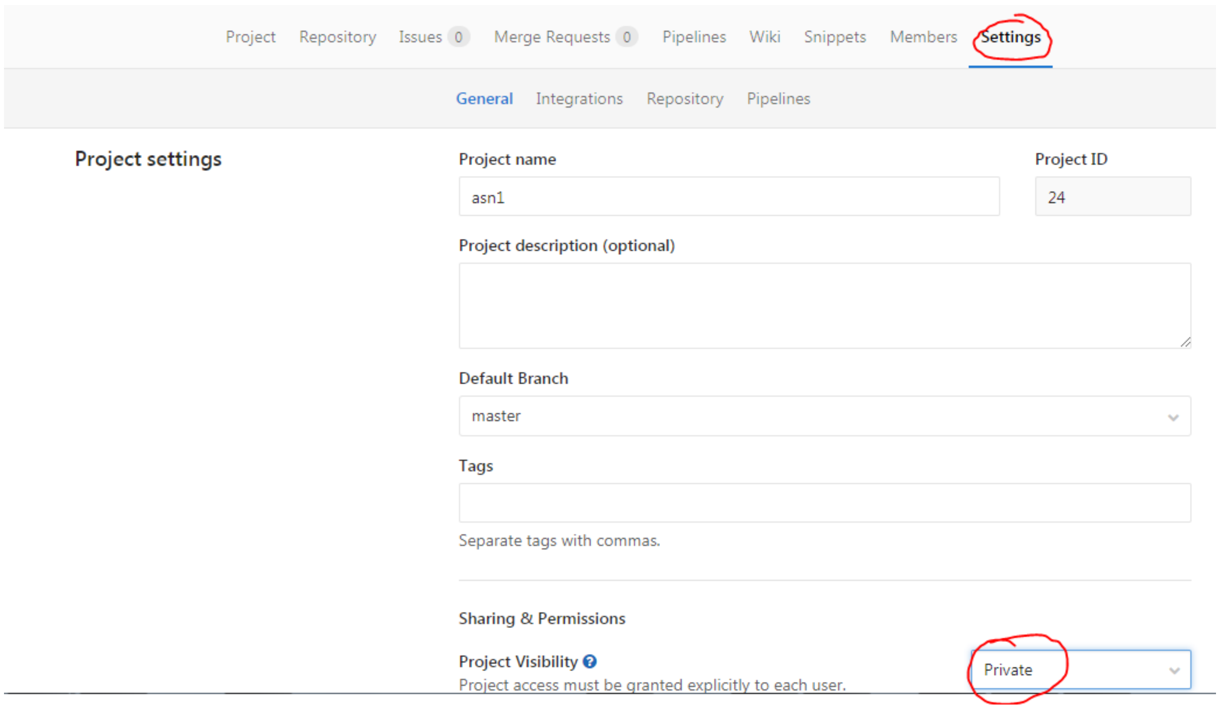
In this assignment, you will:

- Write a simple implementation of a calculator.
- Write unit tests for testing the methods of the calculator.
- Keep track of your progress using version control.
- Use various software engineering tools to help create quality software (static and style analysis, memory leak checking, continuous integration)

Instructions

Setup

1. Go to the Git repository at http://ares-mat17.cs.uleth.ca/gitlab/cpsc2720/asn0. As it is a CS department server, you will only be able to do this on the campus network (or via VPN).
2. Fork the repository so you have your own copy.
3. Set the project visibility for your forked repository to "Private".
   a. This means that other students will not have access to your work.



Project name: asn1

Project ID: 24

Project description (optional)

Default Branch: master

Tags

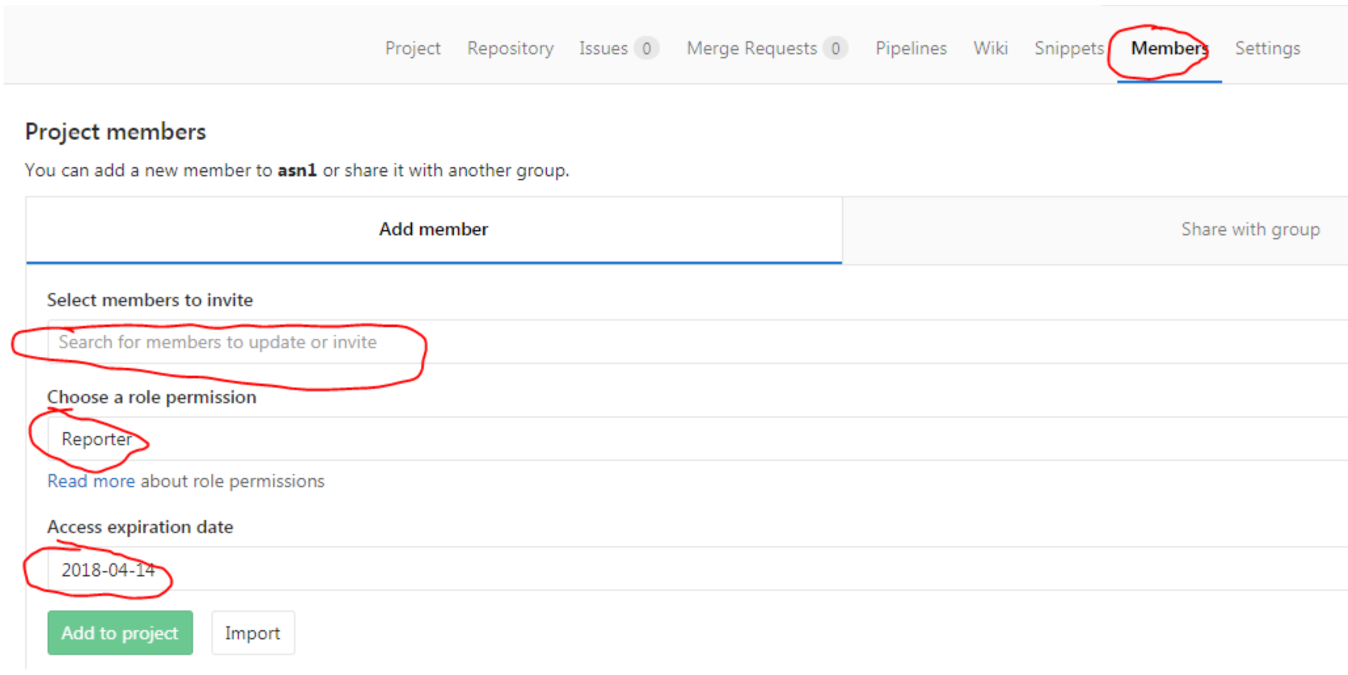
Separate tags with commas.

Sharing & Permissions

Project Visibility: Private

Project access must be granted explicitly to each user.

4. Add the marker and lab instructor as members of your project with the permission "Reporter".
   a. You will be provided with their CS department user name in the lab and/or on Moodle. This is needed so the marker can grade your assignment and the lab instructor can provide assistance.



Project members

You can add a new member to **asn1** or share it with another group.

Select members to invite

Search for members to update or invite

Choose a role permission

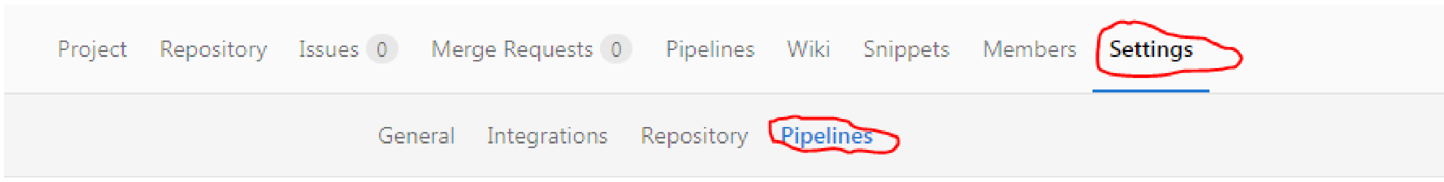
Reporter

Read more about role permissions

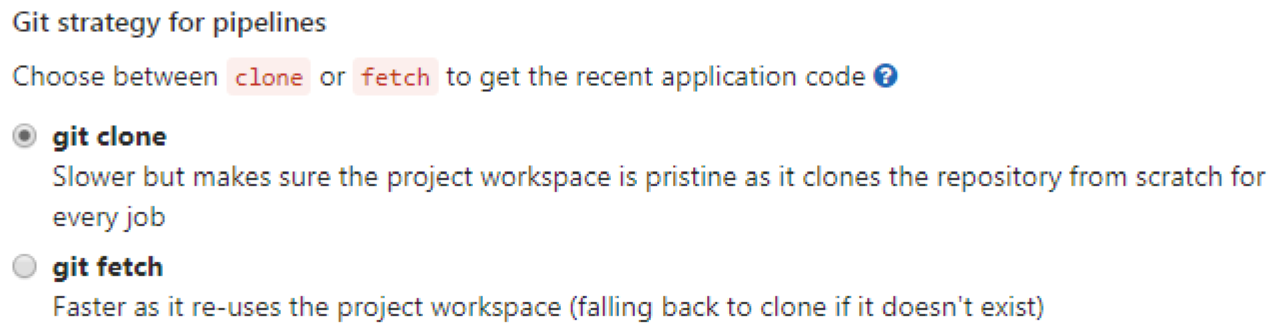
Access expiration date

2018-04-14

5. Setup your GitLab repository for running continuous integration for your project.



a. Set the *Git Strategy* to "`git clone`". You will need to scroll down to find it.



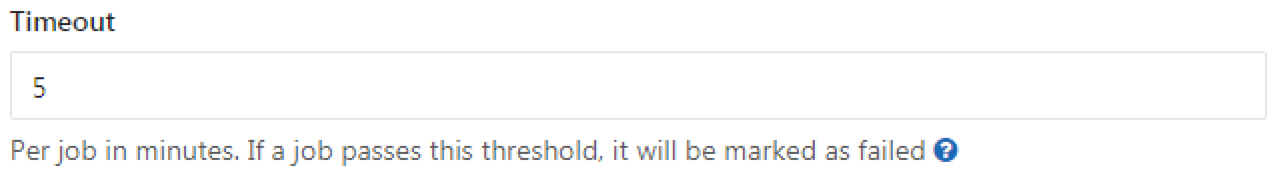
Git strategy for pipelines

Choose between `clone` or `fetch` to get the recent application code

● **git clone**
Slower but makes sure the project workspace is pristine as it clones the repository from scratch for every job

○ **git fetch**
Faster as it re-uses the project workspace (falling back to clone if it doesn't exist)

b. Set the Timeout to 5 (i.e. 5 minutes). Your CI job will be small, so this should be lots of time and will prevent any infinite loops from tying up the CI server or consuming all the disk space.



Timeout

5

Per job in minutes. If a job passes this threshold, it will be marked as failed

Completing the Assignment

1. Create a local clone of your assignment repository.
    a. Run the command `git remote` and verify that there is a remote called `origin`.
        i. `origin` is the link to your repository of GitLab and is where you will be pushing your changes.
2. Open the project in `Code::Blocks.`
3. Build and run the project. It should show "`Running 0 tests from 0 test cases`". If you have problems:
    a. Check the build configuration to confirm that the `GTest` library will be linked in and the `.h` files will be found.
        i. Open the *Build options* for the project.
        ii. Go to the *Linker settings* tab.
        iii. Check that `–lgtest` is in the *Other linker options* textbox. If not add it.
    b. Go to the *Search directories* tab
    c. Check that the *Compiler* tab has the `include` directory where the header files are.
4. Create a class `Calculator` with the following `public` methods using the Test Driven Development technique:
    a. `add(x,y)` : takes in two integers and returns the sum of `x` and `y`.
    b. `sub(x,y)` : takes in two integers and returns the difference between `x` and `y`.
    c. `mult(x,y)` : takes in two integers and returns the product of `x` and `y`.
    d. `div(x,y)` : takes in two integers and returns the quotient of `x` divided by `y`. The method throws a `div_by_zero_error` if `y = 0`.
5. Your header file (`Calculator.h`) is to be in the `include` directory and your implementation (`Calculator.cpp`) is to be in the `src` directory.
6. Write unit tests for all of the public methods (except destructors and constructors for exceptions) in `test/TestCalculator.cpp`.
    a. At this point all your unit tests should fail as the methods have no implementation.
7. Implement the public methods of `Calculator`.

Notes

- A `Makefile` is provided which:
    - Builds a testing executable (`make testCalc`)
    - Checks for memory leaks (`make memcheck`)
    - Runs static analysis (`make static`)
    - Runs style checking (`make style`)
    - Runs all of the checks (`make all`)
- A continuous integration configuration file (`.gitlab-ci.yml`) is provided for you. It is not expected

that you will need to change this file.

Grading

You will be graded based on your demonstrated understanding of unit testing, version control, and good software engineering practices. Examples of items the grader will be looking for include (but are not limited to):

- All public methods of `Calculator.h` are tested by unit tests.
- Version control history shows an iterative progression in completing the assignment. You are expected to have a minimum of four new commits in your repository (i.e. one for each new test case for each method).
- Version control repository contains no files that are generated by tools (e.g. object files, binary files, documentation files)
- Memory leak checking, static analysis and style analysis show no problems with your code.

Submission

There is no need to submit anything, as GitLab tracks links to forks of the assignment repository.

- Make sure that the permissions are correctly set for your repository on GitLab so the grader has access. **You will receive an automatic 0 (zero) for the assignment if the grader cannot access your repository.**