CPSC 2720 – Assignment 1 (Spring 2019)

Overview

In this assignment, you will:

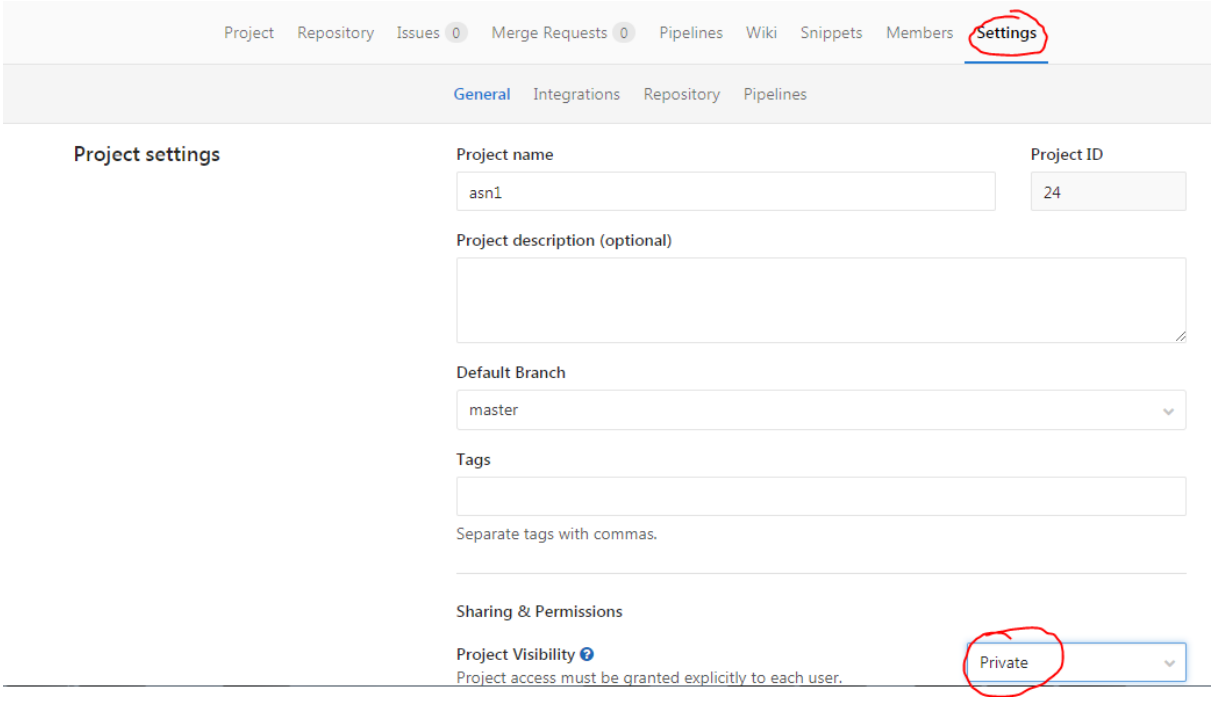
- Write unit tests for a provided library that represents various geometric shapes.
- Keep track of your progress using version control.
- Discover and report bugs that appear in a provided library.

Instructions

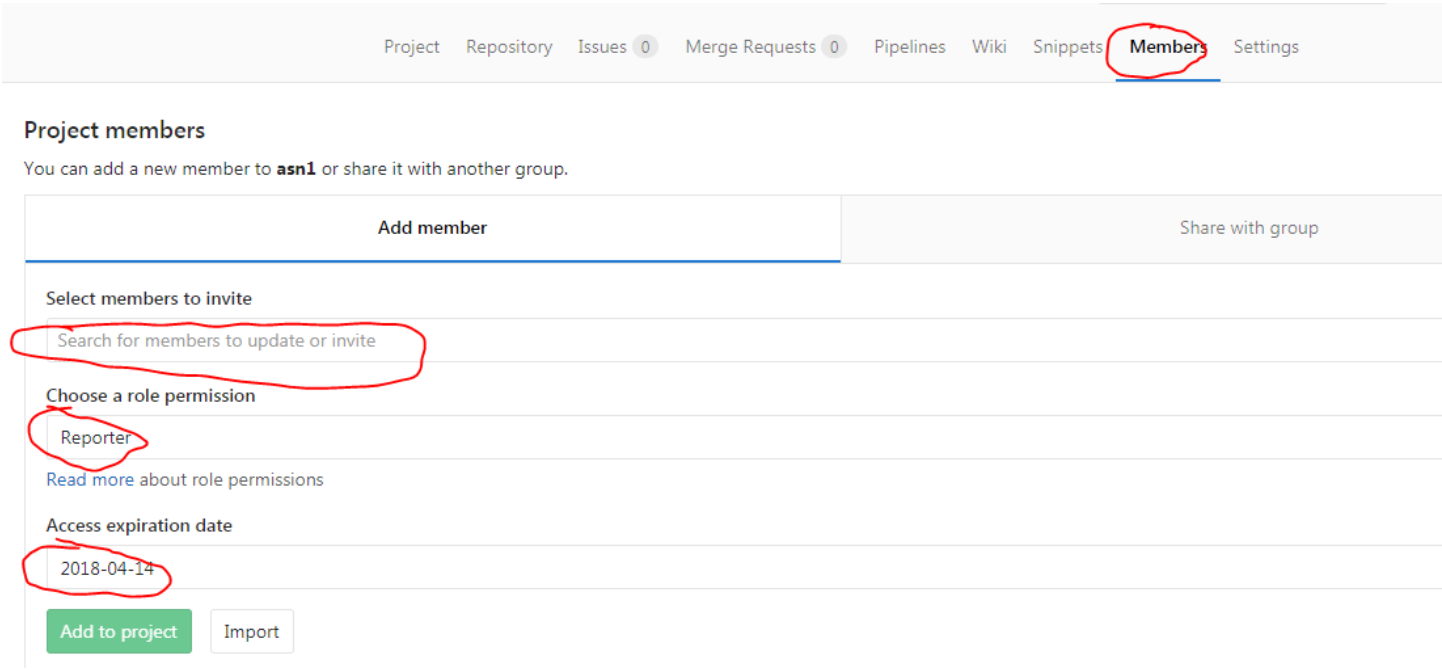
Setup

1. Go to the Git repository at http://ares-mat17.cs.uleth.ca/gitlab/cpsc2720/Geometry/asn1. As it is a CS department server, you will only be able to do this on the campus network (or via VPN).
2. Set your notification settings for this repository to "Watch" so you will receive email notification if there are any changes to repository (e.g. clarifications are added to the instructions).
    a. It is not expected that changes will be needed – this is "just in case"

3. Fork the repository so you have your own copy.
    a. If you do not do this step, the marker will not be able to find your assignment repository and **you will receive an automatic 0 for the assignment.**
4. Set the project visibility for your forked repository to "Private".
    a. This means that **no one else** will not access to your work unless you give them permissions (see the next step).

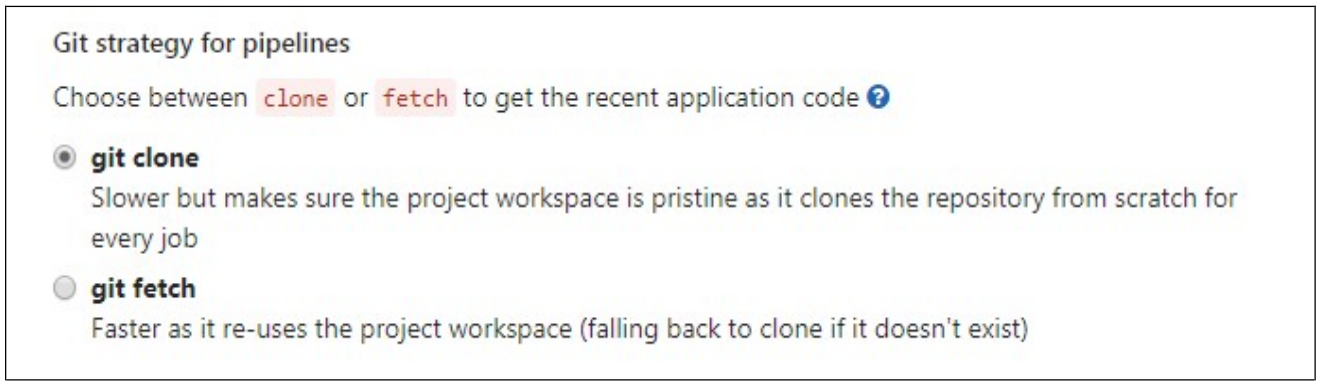


5. Add the marker, lab instructor, and Dr. Anvik as a member of your project with the permission "**Reporter**". You will be provided with their CS department user name in the lab. This is needed so the marker can grade your assignment and the lab instructor or Dr. Anvik can provide assistance.

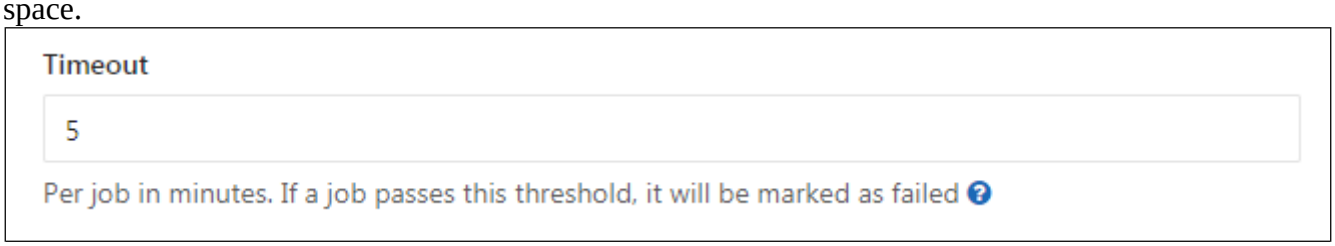


6. Setup your GitLab repository for running continuous integration for your project.

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



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.



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. Compile and run the provided code. One unit test should run, it will fail.

    If there is a problem, check the following settings:

    a. The `gtest` library and the `shapes` library are linked in.
        i. Open the *Build options* for the project.
        i. Go to the *Linker settings* tab.
        i. Add `libshapes-clean.a` to the *Link libraries* window
        iv. Type `-lgtest` in the *Other linker options* textbox
    b. Code::Blocks knows where to find the files.
        i. Open the *Build options* for the project.
        i. Go to the *Search directories* tab
        i. Check that the *Compiler* tab has the `include` directory where the header files are.
        iv. Check that the *Linker* tab contains the project directory where the `libshape-clean.a` file is.

4. Generate the project documentation using `doxygen` which provides an easy to read format for the comments.
    a. Use `make docs` to do this.
5. Read through the generated documentation (or header files) for all of the methods in all of the classes to understand the expected output of the methods.
6. Write unit tests for all of the methods (except destructors and constructors for exceptions) of the concrete classes (e.g. Quadrilateral, Cylinder, Circle, etc.).
    a. Start by writing unit tests that test the "clean" library (`libshape-clean.a`) which is intended to be bug free. This will help to make sure that you don't have any problems with your unit tests.
    b. **There are about 6 intentional implementation bugs in the "buggy" library** (`libshapes-bugs.a`) and your unit tests should identify them.
        i. If a unit test passes for the "clean" library but fails for the "buggy" library, then you can be sure that you found one of the intentional bugs.
        i. **Do not** change any of the `.h` files. You are to write tests according to the specification of the methods given. If you think you found a bug in the "clean" library, create an issue on the assignment repository (http://ares-mat17.cs.uleth.ca/gitlab/cpsc2720/Geometry/asn1).
        i. **Do not** create your own implementation of the class files to make your unit tests pass. Some of your unit test should not pass for the "buggy" library.
    c. For the value of pi (3.1415…) use the constant `M_PI` defined in `<cmath>`.
    d. For this assignment, two floating-point values are considered equal if they are within an error bound of 0.0001 (i.e. 4 decimal places).
    e. Unit tests are to be organized by the class they test (i.e. each class-under-test has a corresponding test fixture file).
    f. You can use http://www.calculatorsoup.com/calculators/geometry-calculators.php (or other online calculators) to determine the expected value for your unit tests.
7. File bug reports in **your repository** (not the assignment repository) for the issues in the `libshapes-bugs.a` discovered by your unit tests. Be specific enough that it will be clear what bug you found.
    a. Try your best to report *unique* bugs. For example, assume that superclass `A` has two subclasses `B` and `C`. Class `A` has the method `foo()`, which is inherited by classes `B` and `C`. If your unit tests reveal the same bug in `B.foo()` and `C.foo()` then the fault is likely in `A.foo()` and should only be reported once (not twice). However, you can indicate in the bug report that the failure occurs in both `B` and `C`.

Notes

- A `Makefile` is provided which:
    - Builds a testing executable using both the "clean" (`make testShapes-clean`) and "buggy" (`make testShapes-bugs`) libraries.
    - 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.
- You may find a bug that cause a test to `SEGFAULT`, which stops the remaining tests from running. You can use `DISABLED_` to skip that test (e.g. `TEST(MyTests, DISABLED_SegFaultTest)`) and continue your testing.

Grading

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

- All public methods of all concrete classes are tested by unit tests.
- Use of equivalence partitioning in the creation of test cases.
- Version control history shows an iterative progression in completing the assignment. You are expected to have a minimum of five new commits in your repository (i.e. one for each new test file).
- 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.
- Bug reports are clear, concise, and describe the problem in such a way that a developer could replicate the problem, and use the format discussed in class.

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.**

Appendix

Updating the Assignment Files

The following information is to be used in the case that the assignment is updated with clarifications or corrections.

1. Create an upstream remote to the original assignment repository from your local repository:

   ```
   git remote add upstream http://ares-mat17.cs.uleth.ca/gitlab/cpsc2720/Geometry/asn1.git
   ```

   This command creates a link from your local repository to the original assignment repository. You will not have permissions to push to the assignment repository, so you will get an error if you try.

2. To get the updates from the assignment repository, you can pull them into your local repository.

   ```
   git pull upstream  master
   ```

    a. If there are any merge conflicts, you will need to resolve them.

3. Commit the changes to your local repository.