CPSC 2720 – Assignment 1

Overview

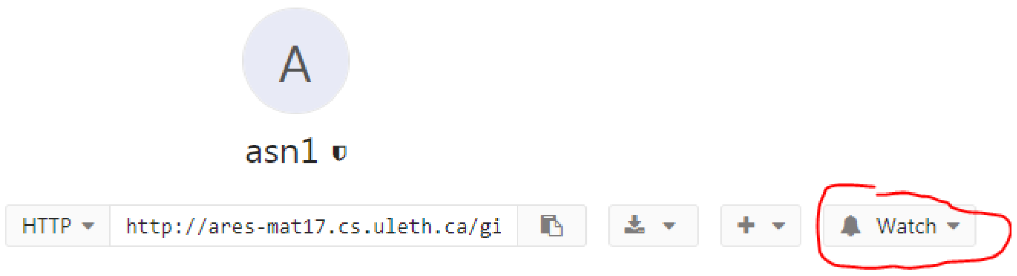
In this assignment, you will:

- Write unit tests for a provided library that represents various geometric shapes.
- Keep track of your progress using version control.
- Report bugs in the library.
- Use various software engineering tools to help create quality software (static and style analysis, memory leak checking, continuous integration)

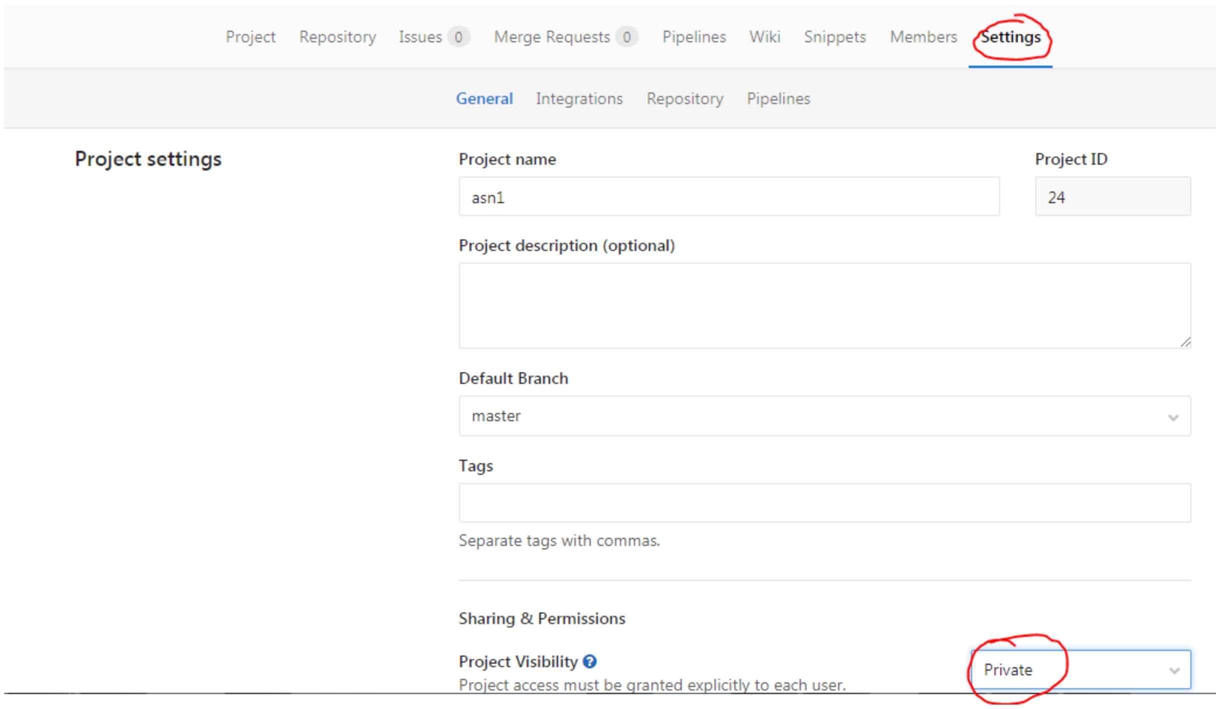
Instructions

Setup

1. Go to the Git repository at http://gitlab.cs.uleth.ca/cpsc2720/GUI/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 these 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.
4. Set the project visibility for your forked repository to "Private".
   a. This means that other students will not access to your work.



Project settings

Project name: asn1

Project ID: 24

Project description (optional)

Default Branch: master

Tags

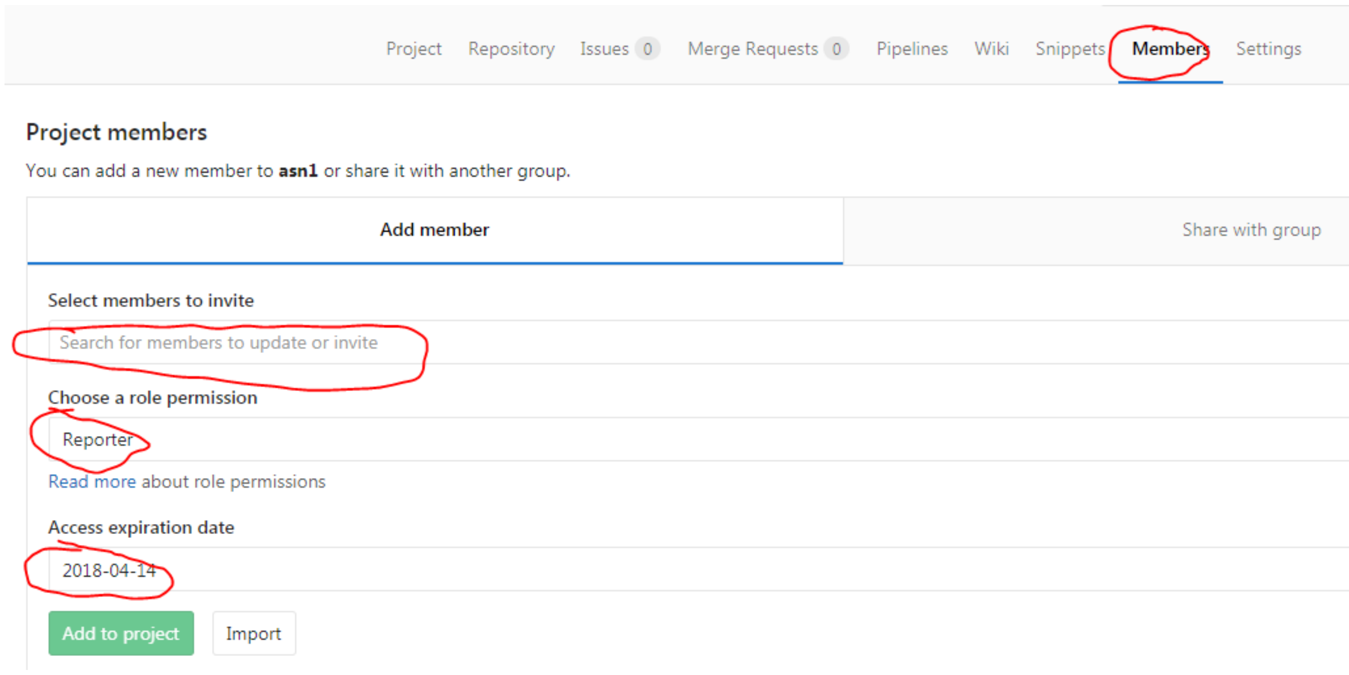
Separate tags with commas.

Sharing & Permissions

Project Visibility: Private

Project access must be granted explicitly to each user.

5. Add the marker (`mark2720`) and lab instructor (`wilsonn`) as members of your project with the permission "`Reporter`".
   a. This permission 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.

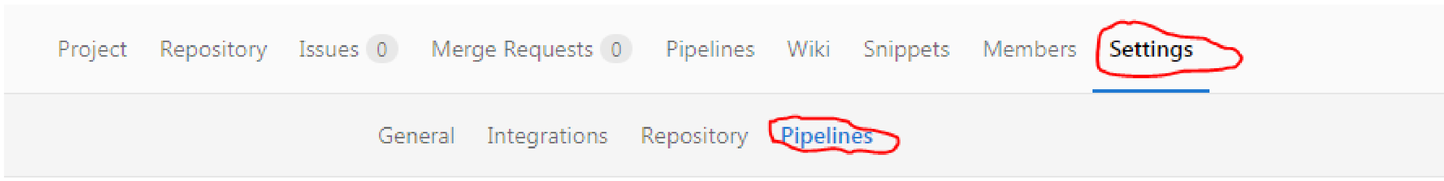
Select members to invite: Search for members to update or invite

Choose a role permission: Reporter

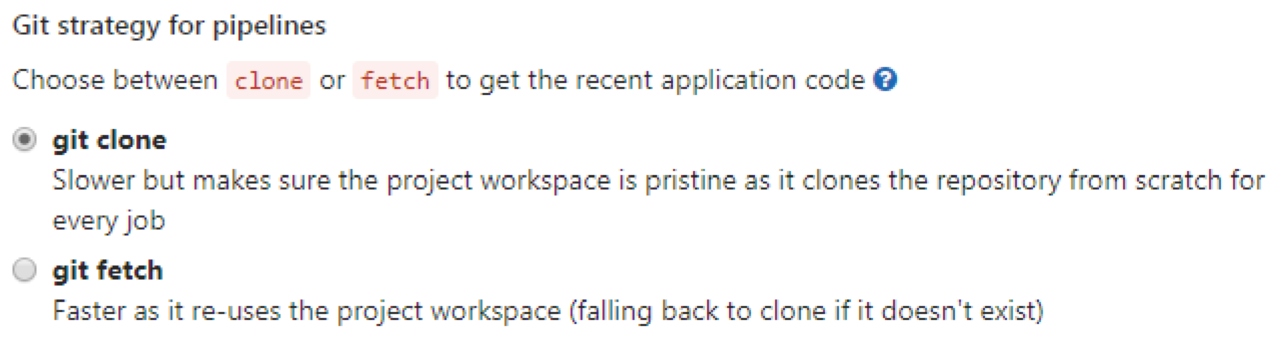
Read more about role permissions

Access expiration date: 2018-04-14

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.



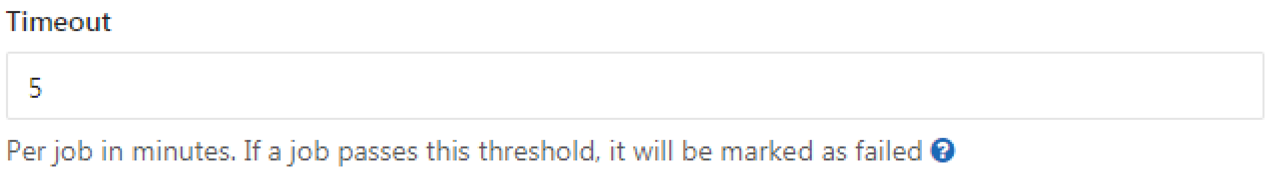
Git strategy for pipelines

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`.
   a. The Code::Blocks project contains links to files that don't exist (e.g. `TestImage.cpp`). These were left in the project file to give you some guidance about naming your test fixtures.
   b. Use `Remove file from project` on each of these missing files to remove them from the project.
3. Compile and run the provided code. A unit test should run, which fails.

   If there is a problem, check the following settings:

   a. The `gtest` library and the `gui` libraries are linked in for the `Clean` and `Buggy` build configurations.
      i. Open the *Build options* for the project.
      ii. Select the `Clean` build configuration.

            1. Go to the *Linker settings* tab.
            2. Check that `libgui-clean.a` is in the *Link libraries* window. If not, add it.
            3. Check that`-lgtest` in the *Other linker options* textbox. If not, add it.
        iii. Select the `Buggy` build configuration.
            1. Go to the *Linker settings* tab.
            2. Check that `libgui-bugs.a` is in the *Link libraries* window. If not, add it.
            3. Check that`-lgtest` in the *Other linker options* textbox. If not, add it.
    b. `Code::Blocks` knows where to find the files.
        i. Open the *Build options* for the project.
        ii. Go to the *Search directories* tab
        iii. 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 `libgui-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 the 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 public methods (except destructors and constructors for exceptions) of the concrete classes (e.g. Coordinate, Image, Textbox, etc.).
    a. Start by writing unit tests that test the "clean" library (`libgui-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 8 unique intentional implementation bugs in the "buggy" library** (`libgui-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 seeded bugs.
        ii. **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://gitlab.cs.uleth.ca/cpsc2720/GUI/asn1).
        iii. **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. Unit tests are to be organized by the class they test (i.e. each class-under-test has a corresponding test fixture file).
7. File bug reports in your repository for the issues in the `lib-guibugs.a` discovered by your unit tests. Be specific enough that it will be clear to the marker 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" (`tests_clean`) and "buggy" (`tests_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.
- For testing `AsciiWindow`, use `diff` to compare a text file(s) created by your unit test(s) with a text file(s) containing the expected output.
    - Example expected output files are provided in `test/output/`, but you may create your own as well.
- 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)`).

Grading

You will be graded based on your demonstrated understanding of:

1. Unit testing,
2. Version control,
3. Bug reporting, and
4. 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://gitlab.cs.uleth.ca/cpsc2720/GUI/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 and 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.
   b. If there is a file with a complicated merge conflict (e.g. a PDF or `.cbp` file)
      i. `git fetch upstream`
      ii. `git checkout FETCH_HEAD <filename>`

3. Commit the changes to your local repository.