Instructions : Complete all steps below to checkout starter code from GitHub, update files and CMake scripts, configure and test your code, and confirm your score with GitHub actions.

Before getting started, you will need a GitHub account and a way to edit repositories. If you do not already have a GitHub authentification token or SSH key setup, a step-by-step guide is available here.

If at any step you run into issues or get confused, scroll to the end of this document for hints and common issues.

Part 1: Getting Started with GitHub

1. **Click on your GitHub Classroom Link, provided to you by your instructor.** If you get an error stating you do not have access to the repository, check your email associated with your Github account. You should have automatically received an email with a link to your repository, and once you click on that link you should have access.

2. **Copy your repository address.** Click the green 'Code' button to find this link. **Make sure the 'HTTPS' button is selected if you are using authentication tokens.**

3. **Clone the repository.** On your local machine, open up a terminal/Command Line. Clone the repository with the following command, where 'URL' is the repository address copied in the previous step:

   ```
   git clone URL
   ```

   If you are asked for a password, use the GitHub authentication token (instructions for this token are linked at the top of this PDF).

4. **Move into the folder you just checked out**. To see all folders and files in your current directory (including the repository that you just checked out), type the following:

   ```
   ls
   ```

   You can enter this folder with the following command, where *foldername* is the name of your new folder.

   ```
   cd foldername
   ```

5. **Pull submodule updates.** Initial code, configuration, and testing environments are provided for you in a separate GitHub repository, linked as a submodule. You cannot change this code. Make sure to pull the latest updates to this submodule with the following:

   ```
   git submodule init
   git submodule update --remote
   ```

Part 2: CMake

CMake is a group of tools for compiling code on any given computer. The library that you cloned in Part 1 must be compiled with CMake. Follow the steps below to complete basic tasks with CMake.

1. **Create a build folder.** All CMake commands should be run within a build folder so that they can be easily cleaned up. Create a new folder in the homework0 repository called *build*

```
mkdir build
```

2. **Change directory into build.** Change the current directory to build

```
cd build
```

3. **Configure the current CMake directory.** Run cmake from within the build directory. The *cmake* command takes, as input, the directory where it can find your CMakeLists.txt file, which is in the parent directory of your build folder

```
cmake ..
```

4. **Compile the codebase.** After the configuration successfully completes, you can compile the library with the command *make*. You should see an error that the linker cannot find the function *return0()*.

```
make
```

5. **Create a C or C++ file within the main repository directory.** Leave the build directory, stepping back one folder and create a new file. The following commands will create a new file called *filename*. Replace *filename* with the name of your new file (you can choose anything you would like). Note, there are many different ways to create new files, such as using *vi*.

```
cd ..
touch filename
```

6. **Edit this File.** Include the header file *src.hpp* and create a new method called *return0* which returns the integer 0. Note there is no *main* method in this file. This is a method within a library, which has already been declared within the header file (located in the submodule folder).

7. **Edit CMakeLists.txt.** Open the file *CMakeLists.txt* from the main directory. A library called *homework* is created near the bottom of this file. Add your file to this library.

8. **Reconfigure and compile the library.** Go back into the build directory and run the configuration and compile commands again (from steps 3 and 4 above).

```
cd build
cmake ..
make
```

Part 3: CMake Testing

Ctest is a testing environment, which will test your code for correctness. These tests are provided for you, within the *tests* directory of the submodule folder.

Cont.

1. **Open the file *unit_tests.cpp*.** This file is within the *tests* folder of the provided submodule. This a simple test that checks correctness of the method you created. If your method does not return 0 as expected, this test will print to standard error, causing the test to fail. Make sure you can understand what is being tested in this file.

2. **Test for Correctness.** To test that your code changes and the unit tests are working, go back to the build folder and recompile your code. Then, run the tests with *make test*. This will run my provided unit tests. You should see tests pass.

```
cd build
make
make test
```

3. **Update Compile Script.** Add all configuration and compilation commands to the file *compile.sh*, **including your submodule commands**. This is the file that GitHub Classroom will use to compile your code. *If you need further direction on this step, check down in hints.*

4. **Retest Correctness.** Delete your build folder and instead run only the following commands from the main repository folder:

```
sh compile.sh
cd build
make test
```

You should again see that you pass the unit tests.

PART 4: COMMITTING AND CHECKING

Once all of your tests are passing, you need to push the code back to GitHub. **For larger tasks, you should do this step even before all tasks are complete as a way to checkpoint your code.**

1. **Pull Repository Updates.** If your local code is not up-to-date with the online repository, you will not be able to push new changes without first pulling.

```
git pull
```

2. **Check Repository Status.** This command will show which files have been changed, added, or deleted since your last pulled code.

```
git status
```

3. **Add Files.** Any files that have been changed or added to your repository can be added with any of the following commands. Note, this stages these changes, but they will still need to be committed and pushed before they appear in your online repository.

   (a) The following will add the file named *filename* to be committed. Replace *filename* with the name of the file that you want to add.

```
git add filename
```

   (b) The following will add all files in your current directory.

Cont.

```
git add *
```

(c) The following will add only files that have previously been added to the repository and have since been updated.

```
git add -u
```

4. **Commit Changes.** After adding updates and new files, you need to commit these changes with the following. Replace *message* with a description of your update.

```
git commit -m message
```

5. **Push Changes.** After an update has been committed and tagged with a message, push your changes to your online repository. If asked for a password, you will need to enter your GitHub authentication token again.

```
git push
```

6. **Check Repository.** Go back to your GitHub repository and refresh the page if needed. You should see your updated code in your repository.
7. **Check GitHub Actions.** Click on the button 'Actions' and find the message you used with your most recent commit. If there is a yellow circle next to your commit, the workflow is still in progress and you will have to wait until it finishes to see if tests pass. If the circle is red, at least one test has failed. If the circle is green, all tests have passed!
8. **Analyze GitHub Actions.** Click on the action, regardless of the circle color. Note:
   (a) The steps Set Up job and Checkout code should work. If either of these has an error, you may need to reach out for additional help debugging it.
   (b) Click on all additional headers. **They may have a white checkmark even if code is not compiled correctly!**

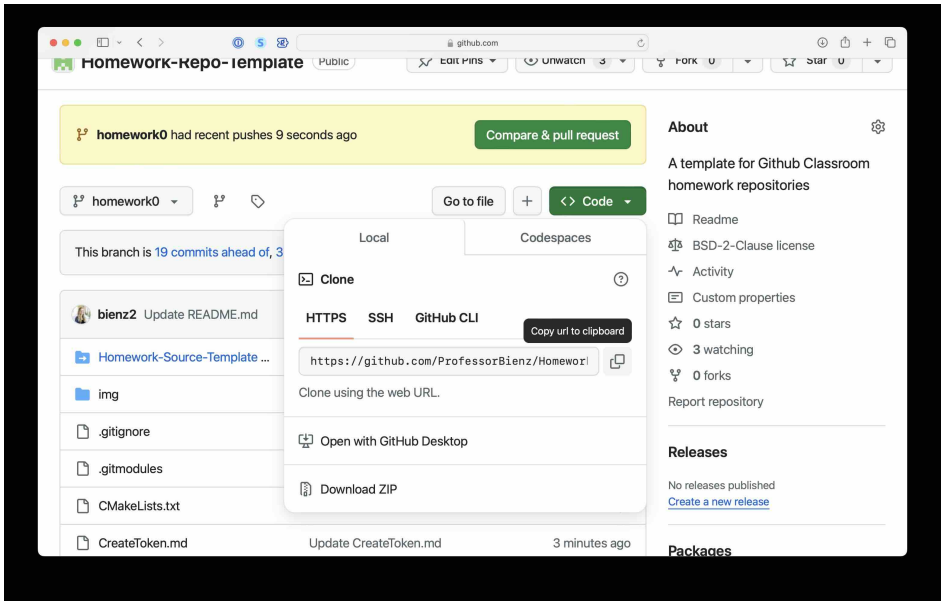
Hints and Common Issues

Included below are hints, including screenshots accomplishing many of the tasks listed throughout this document. At the end of this section, there is also a list of common issues and strategies for solving them.

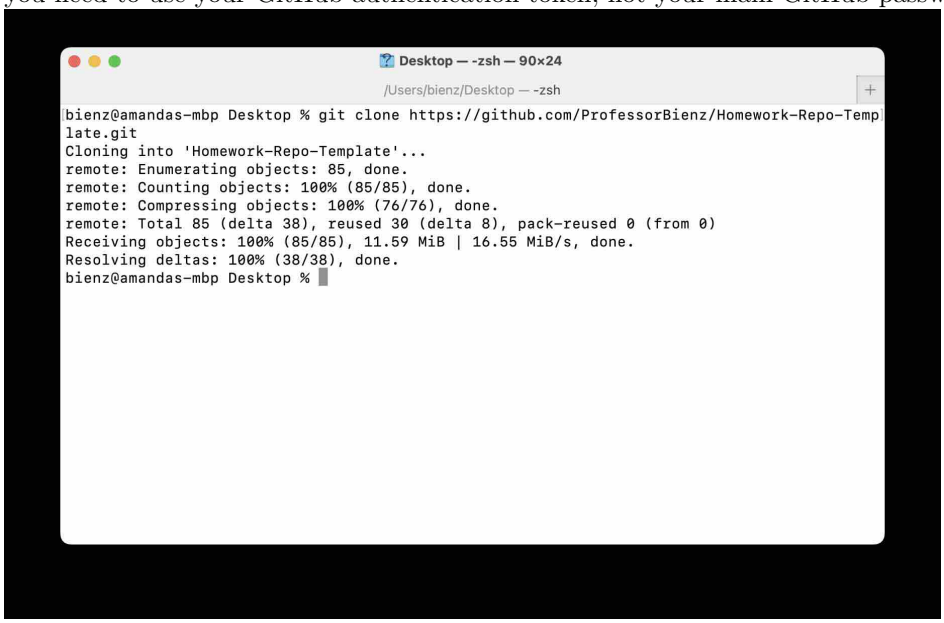
Hints

(1.2:) The image below shows how to find your repository address. This is needed to clone your repository. Copy this address and paste it after 'git clone' for an initial copy of the code.

Cont.

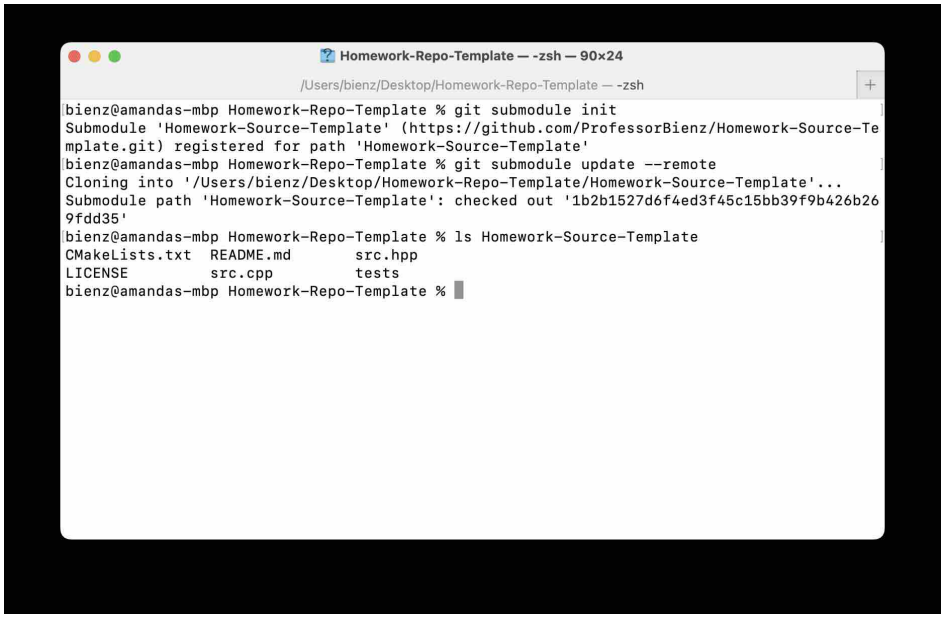


(1.3:) After you clone your repository, you should see similar output. If you get an error, check your that you have correctly copied the HTTPS address from your repository. If you are asked for a password, you need to use your GitHub authentication token, not your main GitHub password.

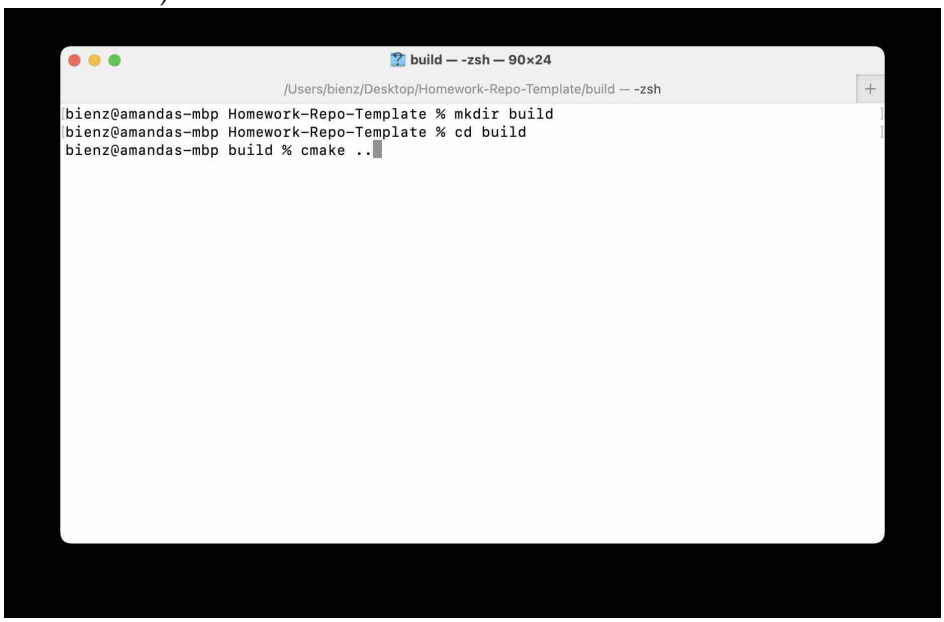


(1.5) After you update your submodule, you may see similar output to the image below. However, if your submodule was already up-to-date, you will see no output when you type this command.

Cont.

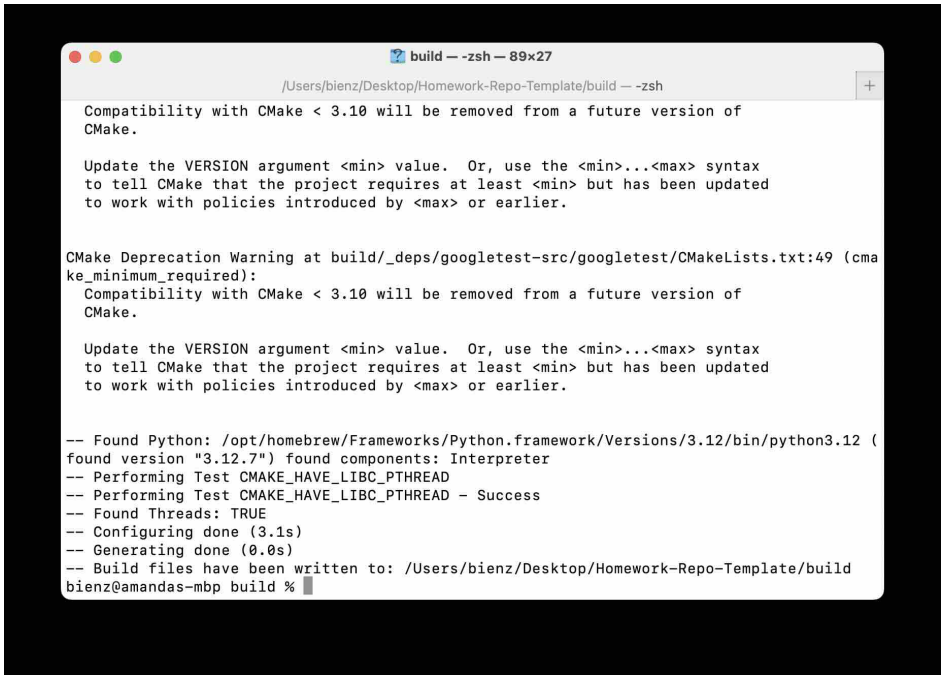


(2.3) The following image shows how to create a new build folder, move into that folder, and run the 'cmake' configuration script from within that folder. **Make sure you have two dots after cmake, pointing to the CMakeLists.txt file, which is in the parent directory of your current location (the build folder).**

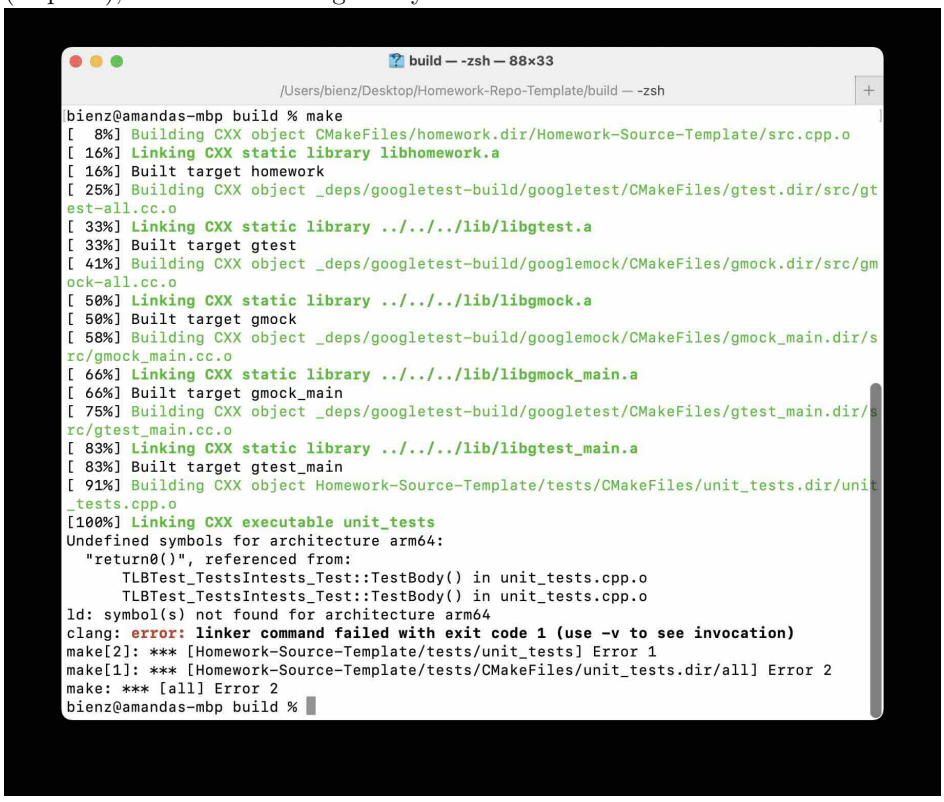


CMake will configure your code, and you should see the line saying 'Build files have been written to...', pointing to a location on your computer. You may see warnings and that is okay, but you should not see any CMake errors.

Cont.



(2.4) When you compile your code, you should see files being built and linked. You should also initially see an error at the end. After you create your new function (step 2.5), and add it to the CMakeLists.txt (step 2.7), this error should go away.



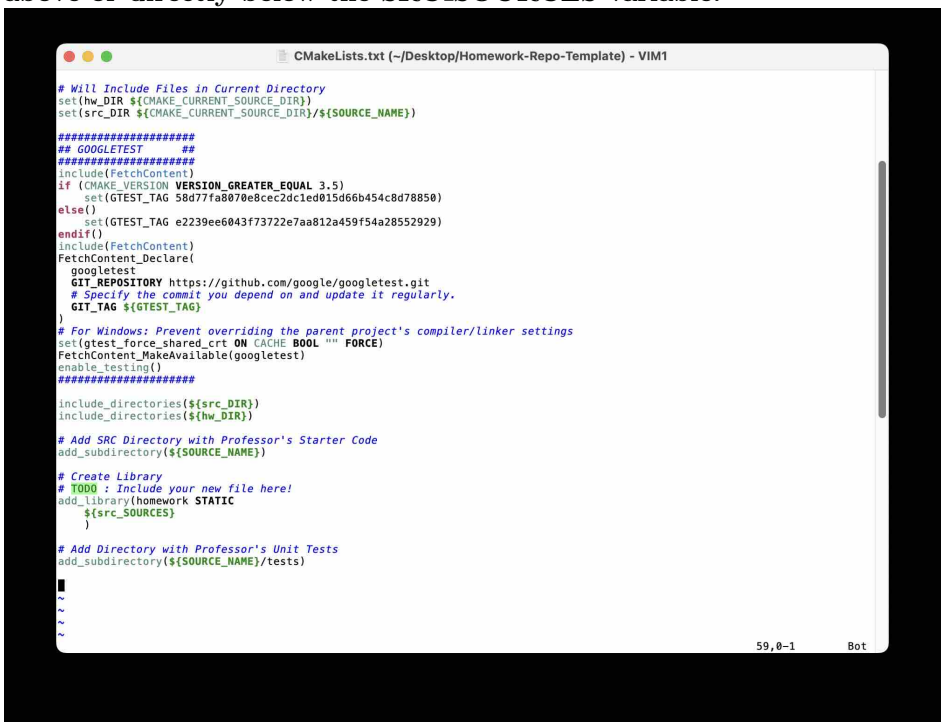
(2.5) There are many options for creating a new file, including:

- 'touch newfilename.cpp' will create a file that you can then edit with the editor of your choice.
- 'vi newfilename.cpp' will create a new file and open it with VI, if installed.

Cont.

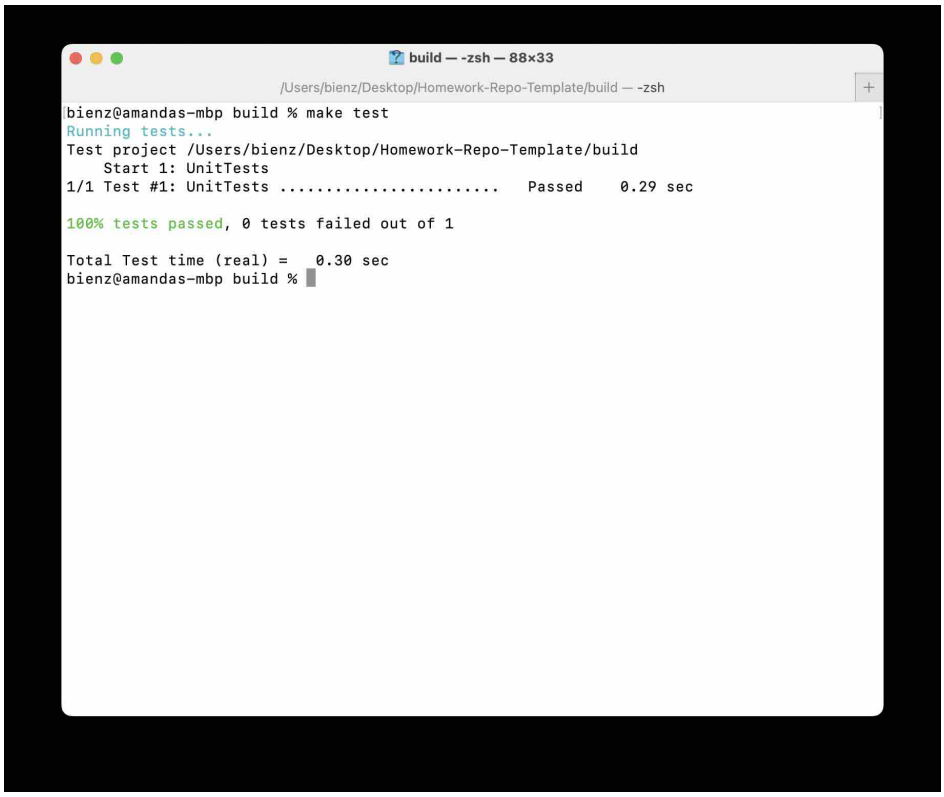
- ‘emacs newfilename.cpp‘ will create a new file and open it with Emacs, if installed.
- Mac users: ‘mvim newfilename.cpp‘ will create a new file and open it with MacVim, if installed, allowing for both standard text editing and VI commands.

(2.7) This is what the CMakeLists.txt looks like. If this is not what you see, you have opened the incorrect file. Make sure you are opening the file CMakeLists.txt that is located within the main directory (not a file by the same name within the submodule). **Add the .cpp file that you created either directly above or directly below the SRC_SOURCES variable.**

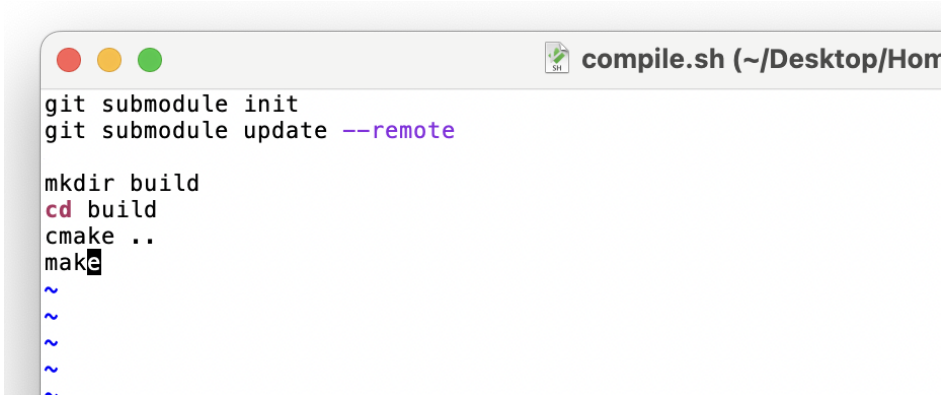


(3.2) Below is what the output of typing ‘make test‘ should look like. If this is not what you see, check out common issues in the subsection below.

Cont.

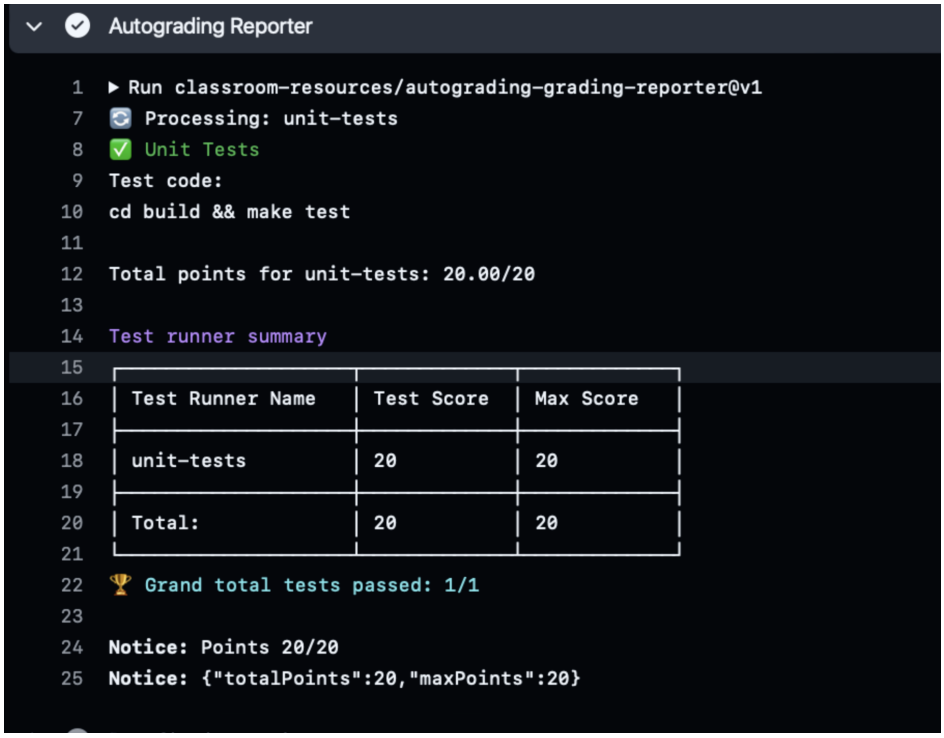


(3.4) This is what your 'compile.sh' script should look like.

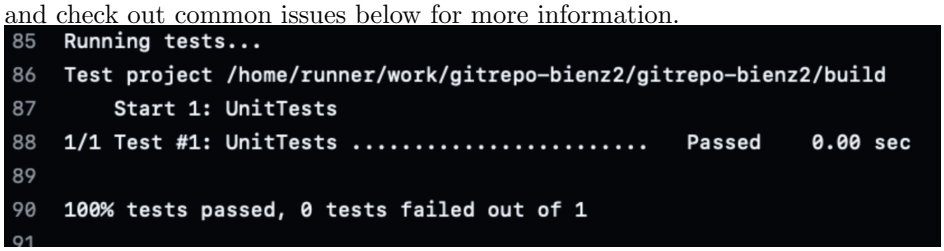


(4.8) This is what you should see if you open up a passing GitHub actions test case (along with a green checkmark next to the Action).

Cont.



If you click on the the dropdown arrow above the Autograding reporter, you should see the following. If your test is not passing, you should click on the drop-down arrow to see why you are not passing, and check out common issues below for more information.



Common Issues

1. **If you get an error while configuring or compiling the library:**
   - If the error says CMake command not found, you need to install CMake.
   - This may be due to an old cache. Try removing everything from within the build directory, and then rerun your configure and compile commands.
   - An error during the command ‘cmake ..‘ means there is an issue with your configuration. Make sure you are running this command from the build directory and that the CMakeLists.txt file is in the parent directory. If you still see an error, double check step 2.7.
   - An error during the command ‘make‘ means there is an issue compiling your code. Make sure you have a C/C++ compiler installed on your computer. If you still see an error, double check step 2.6.
   - If you are Windows, check the CMake on Windows bullet (#6).
2. **If you get an error while running ‘make test‘:**
   - If your see tests similar to the image in the Hints above, check that your method ‘return0‘ returns the number 0.

Cont.

- If you do not see 'UnitTests' running, make sure you are running 'make test' from within your build folder.
- If you still see an error, try running 'make' and double check that your code compiles without error.
- If all else fails, try deleting and recreating your build directory.

3. **If you get an error in your GitHub Actions, click on the dropdown arrow for the individual tests (above your autograding report). Some things you may find:**

   (a) 'CMake Error' will point to the line within the 'CMakeLists.txt' that an error is encountered. If this includes 'add_subdirectory', the issue is likely your submodule. Go back and check that the code from Part 1, step 5 was added to your 'compile.sh' file.

```
60  CMake Error at CMakeLists.txt:48 (add_subdirectory):
61    The source directory
62
63      /home/runner/work/git-cmake-tutorial-bienz2/git-cmake
64
65    does not contain a CMakeLists.txt file.
66
67
```

   (b) 'Cannot find source file' means you are trying to compile code but cannot find the file. Make sure you have added/committed/pushed the new file that you created.

```
64  -- Configuring done (4.7s)
65    Cannot find source file:
66
67      code.cpp
68
69    Tried extensions .c .C .c++ .cc .cpp .cxx .cu .mpp .m
70    .ccm .cxxm .c++m .h .hh .h++ .hm .hpp .hxx .in .txx .
71    .f95 .f03 .hip .ispc
72
73
74  CMake Error at CMakeLists.txt:52 (add_library):
75    No SOURCES given to target: homework
76
```

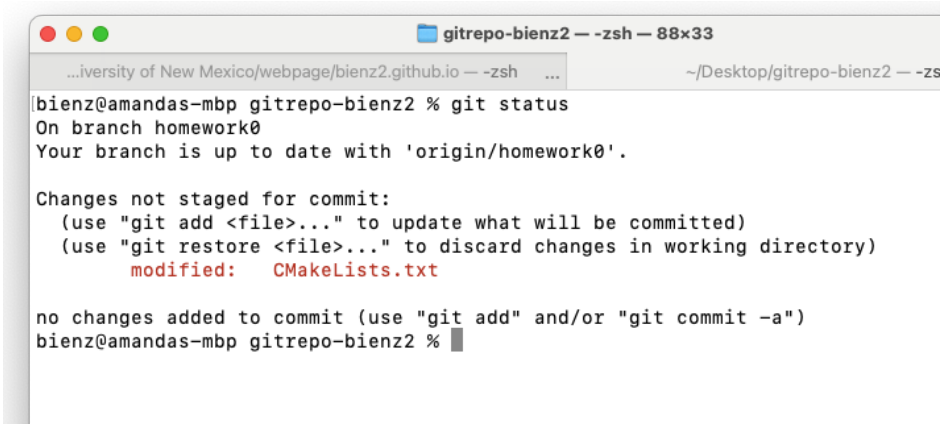
   (c) If your unit test fails, your return0 function does not return the number 0.

```
85  Running tests...
86  Test project /home/runner/work/gitrepo-bienz2/gitrepo-bienz2/build
87      Start 1: UnitTests
88  1/1 Test #1: UnitTests .....................***Failed    0.00 sec
89
90  0% tests passed, 1 tests failed out of 1
91  Errors while running CTest
92
```

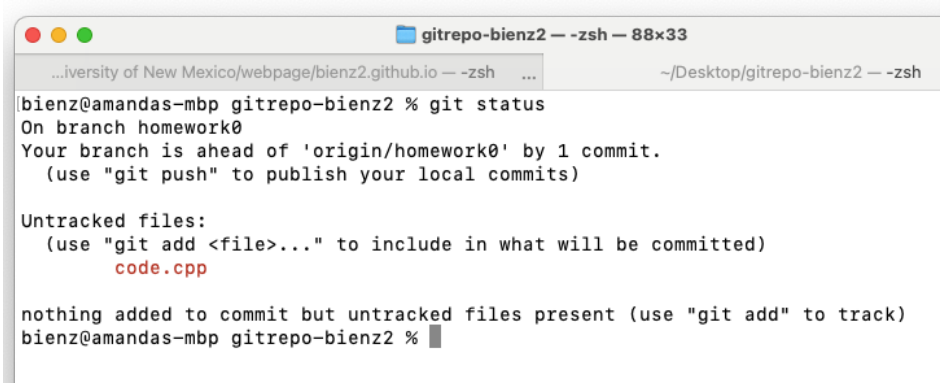
4. **Files Not Pushed.** If you have a file on your computer and it is not showing up in your online repository, there are many possible reasons. Type 'git status' and check if you see any of the following:

Cont.

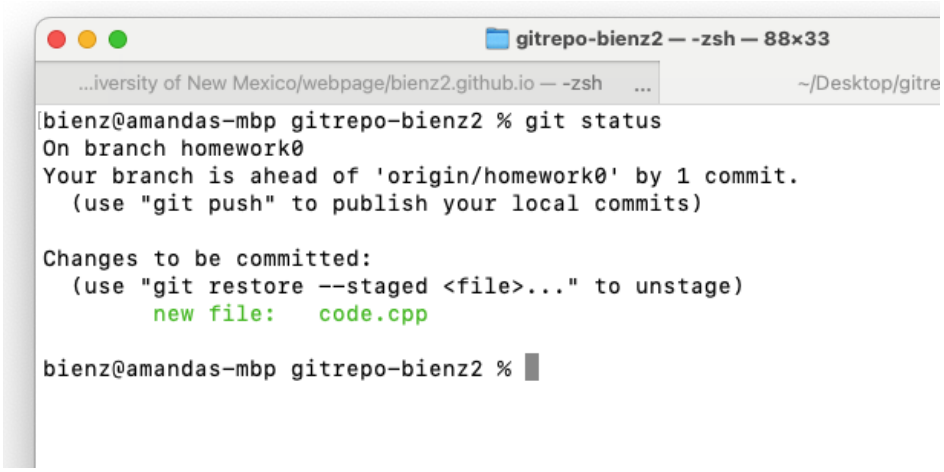
(a) The image below indicates you have changed the file 'CMakeLists.txt'. Add this updated file, commit, and push.



(b) The image below indicates you have created a new file 'code.cpp'. The command 'git add -u' will not add this file, as it is untracked. You need to type 'git add code.cpp'. Then, you will be able to commit and push the file.

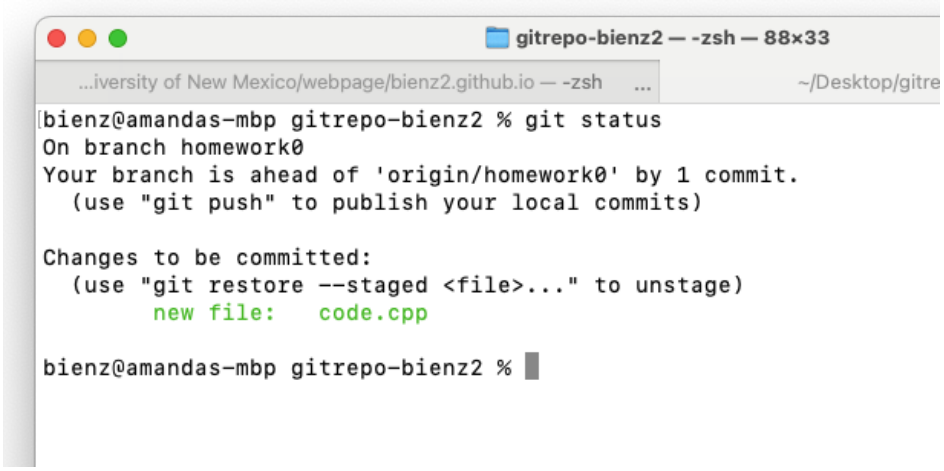


(c) The image below indicates you have already added files, but have not committed or pushed. Commit and push these files.

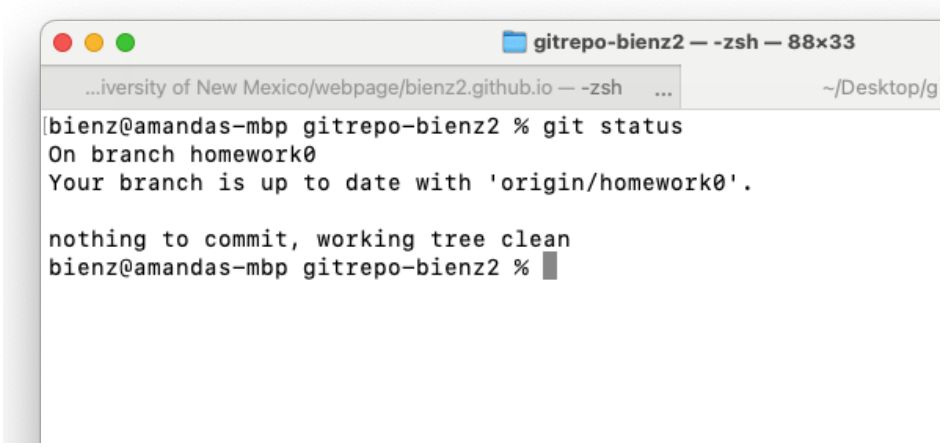


(d) The image below indicates you have already committed files, but have not pushed. Push these files. Hint: there are no colors here. The indication that you have committed is that 'your branch is ahead of origin', where 'origin' is the online repository.

Cont.



(e) If all updated files have been committed and pushed, your repository should look like the following.

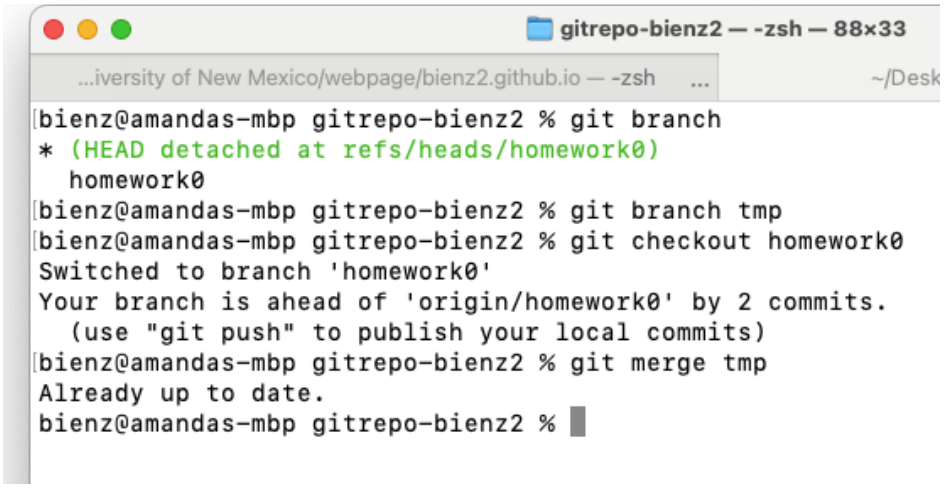


5. **Branch Issues:** If your status shows all files have been pushed, type 'git branch'. This should only show one branch.

(a) If instead you see HEAD detached, this means you are no longer appropriately connected to your GitHub repository.

(b) To fix this, you can create a new branch and merge it into the correct branch, as shown in the following image.

Cont.



6. **CMake on Windows:** If you are using a Windows machine and get an error when configuring with 'cmake ..', try the following:

   (a) If you see an error 'nmake' '-?' failed with: no such file or directory, CMake is unable to find the Make program on your computer. Try adding the following between 'cmake' and '..':

   ```
   -DCMAKE\_MAKE\_PROGRAM=mingw32-make -G "MinGW Makefiles"
   ```

   (b) If you see an error 'CMAKE_CXX_COMPILER not set', CMake is unable to determine which C compiler it should use. Try adding the following between 'cmake' and '..':

   ```
   -DCMAKE\_CXX\_COMPILER=g++
   ```

   Add 'DCMAKE_CXX_COMPILER=g++' between 'cmake' and '..'

   (c) If you see an error 'CMAKE_C_COMPILER not set', CMake is unable to determine which C compiler it should use. Try adding the following between 'cmake' and '..':

   ```
   -DCMAKE\_CXX\_COMPILER=gcc
   ```

   If you see all three errors above, your CMake configure line would be

   ```
   cmake -DCMAKE_CXX_COMPILER=g++ -DCMAKE_CC_COMPILER=gcc
   -DCMAKE_MAKE_PROGRAM=mingw32-make -G "MinGW Makefiles" ..
   ```

7. **Make on Windows** If you run into issues on Windows where the flags from the previous step allowed you to configure (CMake passed) but you get an error that 'make' command not found, try the following command instead

   ```
   mingw32-make
   ```

The End.