ICS 46 Lab Manual

Important Links

Gitlab instance: https://gitlab-ics46-f22.ics.uci.edu (If you are off campus, you will need to use the UCI VPN)

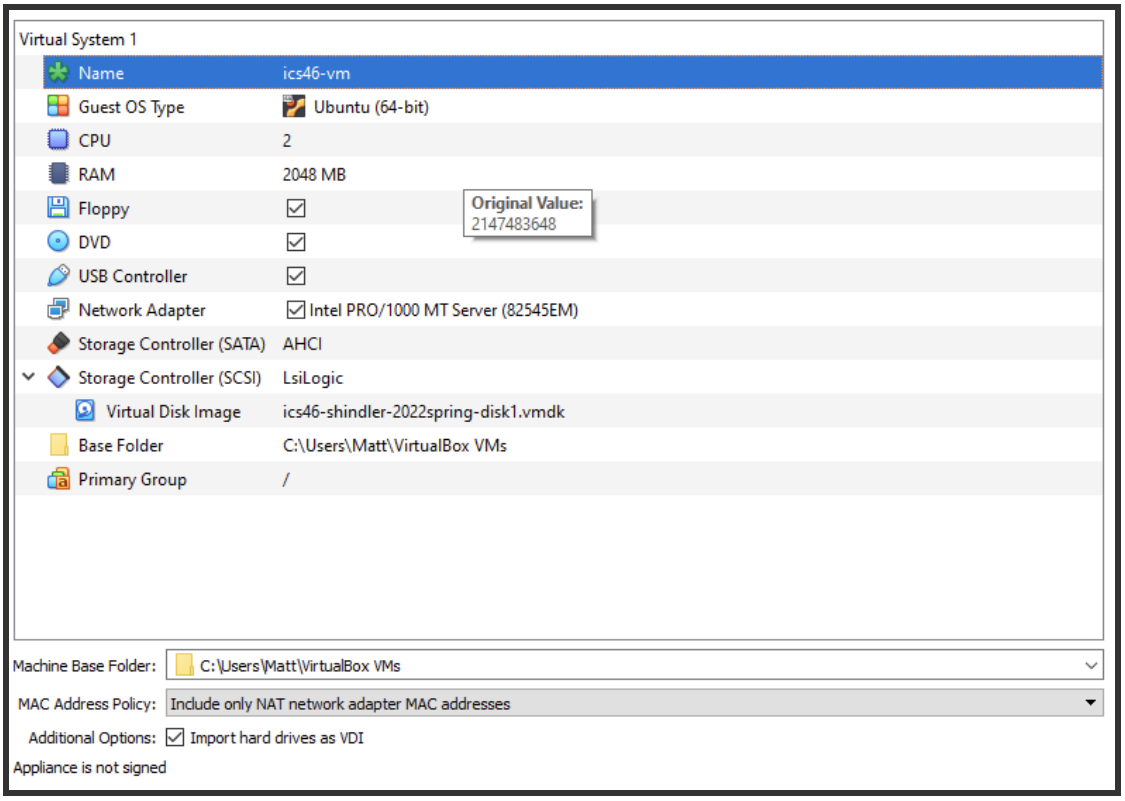
ICS 46 Virtual Machine: https://www.ics.uci.edu/~mikes/ics46/vm/ics46-f22.ova

ICS 46 Virtual Machine (M1): https://www.ics.uci.edu/~mikes/ics46/vm/ics46-f22-arm-vm.tar.gz

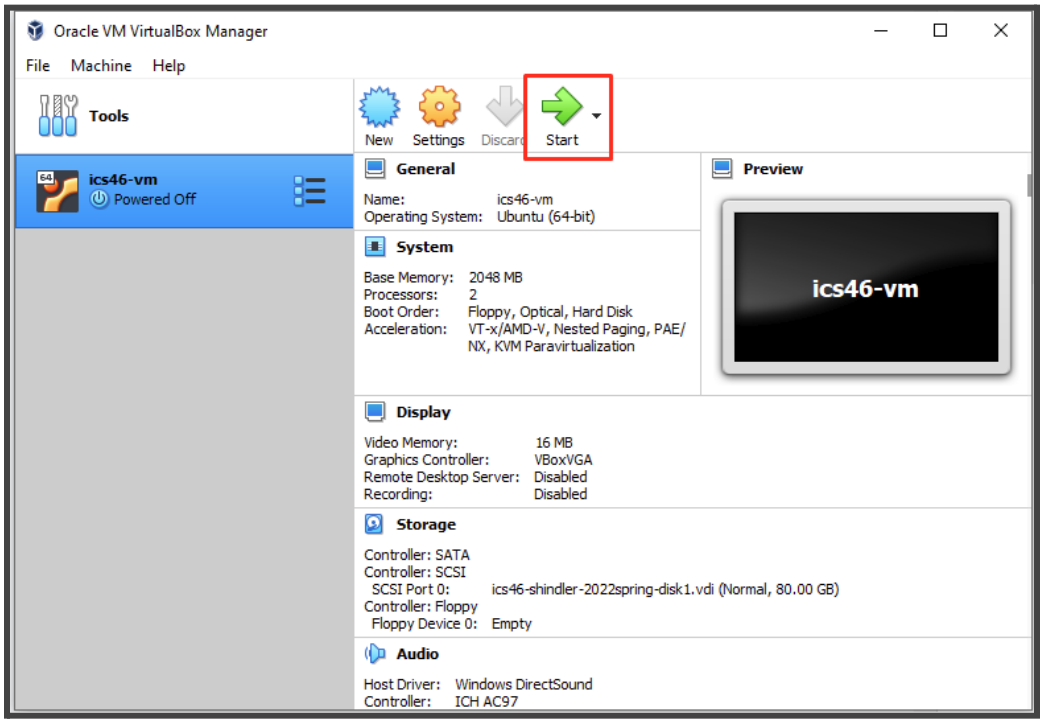
Virtual Machine Setup

Windows / Linux / macOS (Intel Processor only)

1. First download the x86 VM (https://www.ics.uci.edu/~mikes/ics46/vm/ics46-f22.ova).
2. Download and install VirtualBox for your operating system.
3. For macOS, you may need to go to give VirtualBox permissions in
   System Preferences > Security & Privacy
4. Run VirtualBox.
5. Click “Import” (Ctrl + I for Windows/Linux and CMD + I for macOS)
6. Click the folder icon and choose the downloaded ICS 46 virtual machine (.ova file).
7. Make sure the import settings look like the following. I changed the name of the VM and the number of CPUs granted to the virtual machine. How many CPUs you dedicate to the VM will depend on the hardware on your host machine.



8. Click “Import” to start the importing of the VM. This should take a few minutes.
9. Once imported, click the imported VM on the left and then click ‘Start”.



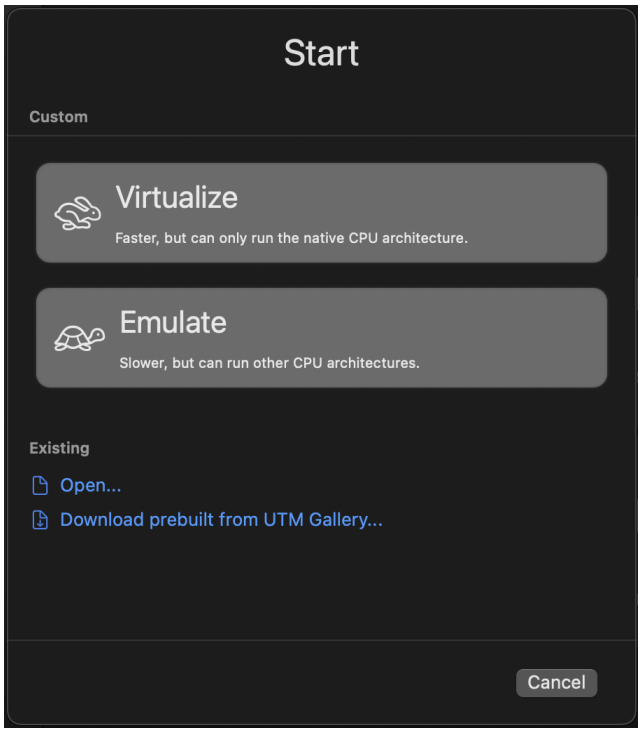
10. You should now be prompted with the login screen. The username *and* password are ics46.

macOS (M1)

1. Download the ARM version of the VM https://www.ics.uci.edu/~mikes/ics46/vm/ics46-f22-arm-vm.tar.gz.
2. Unzip the download.

```
tar -xzf ics46-f22-arm-vm.tar.gz
```

3. VMWare and VirtualBox don't support the M1 yet. You'll need to use UTM https://mac.getutm.app/, which is a GUI wrapper around QEMU, which does have support for the M1 processor. Download and install UTM from the provided link.
4. Run **UTM.**
5. Click the "+" button to create a new virtual machine.
6. At the following screen, select "Open…" at the bottom.



7. Navigate to the extracted `.utm` file that was downloaded and untarred. Select "Open".
8. You should now be prompted with the login screen. The username *and* password are ics46.

Starting a Project

Git Introduction

This course will be using git for managing the distribution and collection of programming assignments. Git is a very powerful distributed version control system that is used heavily in industry. We ***strongly recommend*** getting as familiar with git as possible in this course, as you will most likely use it in some form or another during your post-undergrad job.

Git should already be installed on the VM. Check this by opening the terminal and typing

```
git –version
```

Read the following link for an outline of the git commands https://rogerdudler.github.io/git-guide/. This will give you access to the git command-line interface (and optionally the GUI if you so choose). We recommend using the command line interface to get a deeper feel for git workflow before moving to the GUI.

The above link also includes a primer on git. You should read through this in its entirety. Make sure you understand the basic ideas of **repository, commit, pulling** and **pushing** before continuing, though the following instructions will detail how to use the commands within the context of the class.

Git vs. Gitlab

Though `git` can be used in a purely *local* fashion (i.e. you can create repositories and manage their contents locally), it is most commonly used with a *remote server*. This permits the seamless support for *version control* of the software. Currently, the two main git *platforms* are **Github** and **Gitlab**. While there are many differences between the two, the most important difference for this class is that Github is owned by Microsoft and Gitlab is an open-source software project. UCI ICS provides Gitlab instances for use by classes, so that is what we will be using.

An important distinction is that *git* is the open source program that is used to manage repositories themselves, for example using commands like `git commit`, while *Gitlab* is the *remote git platform* that you will use to push your changes to. This serves as a secondary backup of your code with additional features such as forking, comments, and issues that may be used in the course.

Forking the template repository

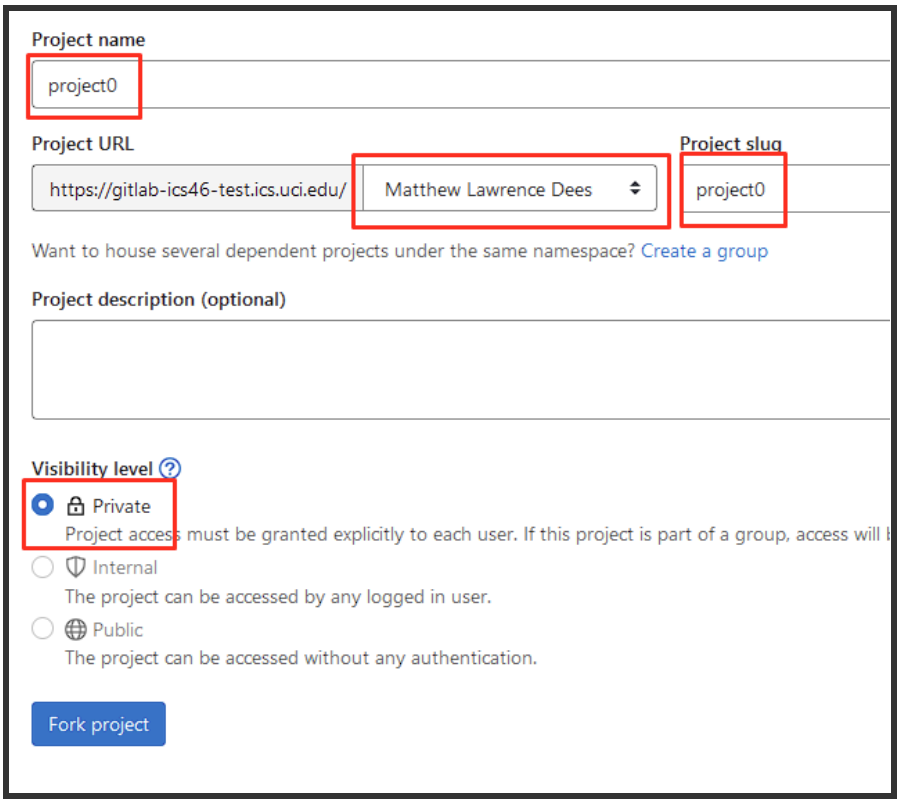
For each project, a *template repository* will be created on the Gitlab instance here: https://gitlab-ics46-f22.ics.uci.edu/mikes. If you are located off campus, you will need to connect to the UCI VPN (https://www.oit.uci.edu/services/security/vpn/) to access the server. For example, the template repository for Project 0 will be located at https://gitlab-ics46-f22.ics.uci.edu/mikes/project0.

You will be responsible for ***forking*** this repository. A *fork* of a repository is effectively an exact clone of someone else's repository. The fork allows you to perform software development on your own copy of the repository without affecting the original (sometimes called ***upstream***) repository. Forking repositories is a very common workflow in open-source software development.

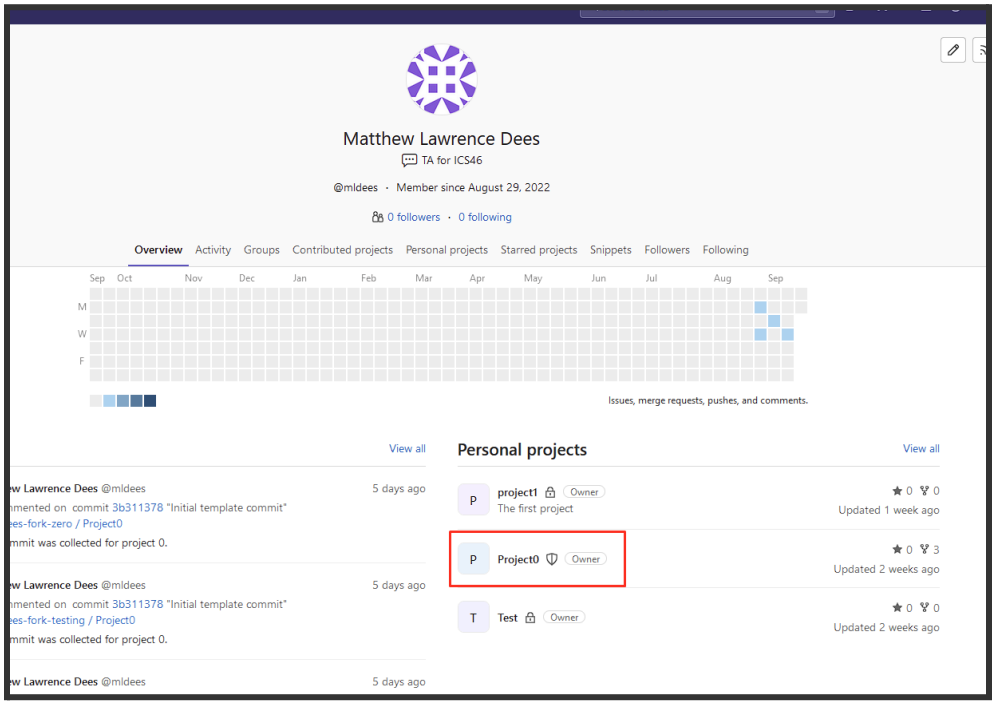
For example, this is how you would fork Project 0:

1. Go the project template link: https://gitlab-ics46-f22.ics.uci.edu/mikes/project0 (for Project 0)
2. You will be asked to sign in with your UCI **ICS** account information (use LDAP).
3. Click the fork button below

4. Make sure the following fields are correct. ***DO NOT CHANGE THE PROJECT NAME***. For Project 0 the project name should be "project0". This should be the default. Make the namespace is set to your personal ICS namespace. Here mine is "Matthew Lawrence…". Make sure the Visibility level is **Private**.



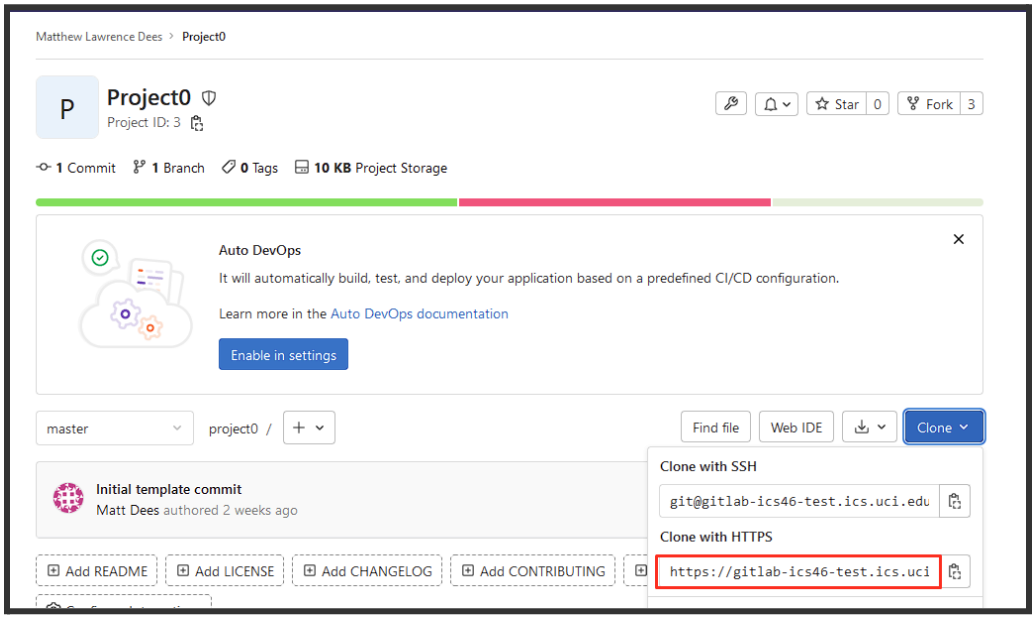
5. Click "**Fork project**".
6. Confirm you have a personal copy of the project under your profile (https://gitlab-ics46-f22.ics.uci.edu/<ICS UCI ID>):



7. You now have a copy of the template repository *stored on the Gitlab server.* The next section will show you how to obtain a copy of it on your machine.

Cloning your copy of the repository

1. Navigate to your copy of the project repository (URL: https://gitlab-ics46-f22.ics.uci.edu/<UCI ICS ID>/project0 for project0).
2. Click "**Clone**" and copy the **Clone with HTTPS** link: (Note: this picture was taken using a test server, the URL `gitlab-ics46-test` will be replaced with the link for the quarter).



3. You can use this URL to clone the repository using the command line in the VM:

   ```
   git clone https://gitlab-ics46-f22.ics.uci.edu/<UCInetID HERE>/project0.git
   ```

   Once you run this command you should be prompted to login with your UCI ICS username and password. A successful clone looks like the following.

```
ics46@ics46-shindler-2022spring : ~ $ git clone https://gitlab-ics46-test.ics.uc
i.edu/mldees/project0.git
Cloning into 'project0'...
Username for 'https://gitlab-ics46-test.ics.uci.edu': mldees
Password for 'https://mldees@gitlab-ics46-test.ics.uci.edu':
remote: Enumerating objects: 19, done.
remote: Total 19 (delta 0), reused 0 (delta 0), pack-reused 19
Unpacking objects: 100% (19/19), 4.60 KiB | 2.30 MiB/s, done.
```

Making your first code change

After cloning your personal fork of the project in the previous section, you should have a local project that looks like this:

```
ics46@ics46-shindler-2022spring : ~/project0 $ ls -l
total 44
drwxrwxr-x 2 ics46 ics46 4096 Sep 19 23:10 app/
-rwxrwxr-x 1 ics46 ics46  530 Sep 19 23:10 build*
-rwxrwxr-x 1 ics46 ics46  119 Sep 19 23:10 clean*
-rw-rw-r-- 1 ics46 ics46 1836 Sep 19 23:10 CMakeLists.txt
-rwxrwxr-x 1 ics46 ics46  322 Sep 19 23:10 debug*
drwxrwxr-x 2 ics46 ics46 4096 Sep 19 23:10 exp/
-rwxrwxr-x 1 ics46 ics46 1723 Sep 19 23:10 gather*
drwxrwxr-x 2 ics46 ics46 4096 Sep 19 23:10 gtest/
-rw-rw-r-- 1 ics46 ics46  134 Sep 19 23:10 memcheck.supp
-rwxrwxr-x 1 ics46 ics46  132 Sep 19 23:10 require*
-rwxrwxr-x 1 ics46 ics46  857 Sep 19 23:10 run*
```

The structure of each project will be explained in a future section. In this section we will explain how to **commit** and **push** a change to your project via the Git ecosystem.

The staging area

A very important concept when using git is the ***staging area.*** This is where git manages changes to your repository before a *commit* happens. You can view the staging area by typing

`git status`

```
ics46@ics46-shindler-2022spring : ~/project0 $ git status
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean
```

Note the message “nothing to commit, working tree clean”. Since I haven’t made any changes to the repository, the staging area isn’t very interesting. Let’s make a change.

As an example, project 0 in ICS46 requires changes to the **app/proj0.cpp** file. Let’s take a look:

```
ics46@ics46-shindler-2022spring : ~/project0 $ cat app/proj0.cpp
#include <unordered_map>
#include <string>
#include "proj0.hpp"


bool verifySolution(std::string s1, std::string s2, std::string s3, const std::u
nordered_map<char, unsigned> & mapping)
{
    return false; // FYI, this stub is not a correct solution.
}
```

Let's use the `sed` command to change the `verifySolution` function to return `true` instead of `false`. Obviously this would not warrant a good submission of project 0, but it will help illustrate the git workflow.

```
sed -i 's/false/true/g' app/proj0.cpp
```

```
ics46@ics46-shindler-2022spring : ~/project0 $ sed -i 's/false/true/g' app/proj0
.cpp
ics46@ics46-shindler-2022spring : ~/project0 $ cat app/proj0.cpp
#include <unordered_map>
#include <string>
#include "proj0.hpp"


bool verifySolution(std::string s1, std::string s2, std::string s3, const std::u
nordered_map<char, unsigned> & mapping)
{
    return true; // FYI, this stub is not a correct solution.
}
```

When working on project 0 you would probably use something like Visual Studio Code to make changes to your project files. Don't worry if you don't understand the `sed` command, it is just a simple way to illustrate a point.

Now, if we run `git status` again, we will see that the **proj0.cpp** file has been modified!

```
ics46@ics46-shindler-2022spring : ~/project0 $ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   app/proj0.cpp

no changes added to commit (use "git add" and/or "git commit -a")
```

We can use the `git diff` command to inspect our change(s).

```
ics46@ics46-shindler-2022spring : ~/project0 $ git diff app/proj0.cpp
diff --git a/app/proj0.cpp b/app/proj0.cpp
index 2af89de..ac651eb 100644
--- a/app/proj0.cpp
+++ b/app/proj0.cpp
@@ -5,7 +5,7 @@

 bool verifySolution(std::string s1, std::string s2, std::string s3, const std
unordered_map<char, unsigned> & mapping)
 {
-    return false; // FYI, this stub is not a correct solution.
+    return true; // FYI, this stub is not a correct solution.^M
 }
```

This looks about right. Once you are happy with your changes, you will need to *add the corresponding files to the staging area.* **Only files that have been added to the staging area will be committed.**

We can add a file to the staging area using the `git add` command.

```
ics46@ics46-shindler-2022spring : ~/project0 $ git add app/proj0.cpp
ics46@ics46-shindler-2022spring : ~/project0 $ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   app/proj0.cpp
```

Committing your changes

The closest analogy to a ***commit*** is a *snapshot*. It is reasonably correct to think of a commit as taking a snapshot of the *files added to the staging area*. In the previous section we added **proj0.cpp** to the staging area. We will now create a commit with this updated **proj0.cpp** file (note: git commits save the entire contents of each staged file at each commit and does not store deltas like older version control technology like Subversion).

Each commit has an associated file snapshot, author, datetime, and message. We first need to configure the author field. You only have to do this once.

```
git config --global user.email EMAIL
git config --global user.name NAME
```

```
ics46@ics46-shindler-2022spring : ~/project0 $ git config --global user.email "mdees@uci.edu"
ics46@ics46-shindler-2022spring : ~/project0 $ git config --global user.name "Matt Dees"
```

Now, we can use the `git commit` command.

When running this command a text editor will open where we can modify the message that will be associated with this commit. For simplicity, we recommend using the `-m` flag to the `commit` command. If you do not provide this flag, git will open a text editor for you to modify the message in.

You can change the git editor using:

```
git config --global core.editor emacs
```

If you're one of *those* people.

Once you save and exit your commit message you will see the following:

```
ics46@ics46-f22:~/project0$ git commit -m "Make verifySolution() more positive"
[master 51ea081] Make verifySolution() more positive
 1 file changed, 1 insertion(+), 1 deletion(-)
```

You can use `git log` to view a history of past commits:

```
ics46@ics46-shindler-2022spring : ~/project0 $ git log
commit f3c613ecae7eb7f94867cf1fc2fb03cc6e995ad8 (HEAD -> master)
Author: Matt Dees <mdees@uci.edu>
Date:   Mon Sep 19 23:50:10 2022 -0700

    Made verifySolution() more positive

commit 3b311378668f781b40fb8272b597422192dc5c1e (origin/master, origin/HEAD)
Author: Matt Dees <mdees@uci.edu>
Date:   Wed Aug 31 20:37:13 2022 -0700

    Initial template commit
```

What's interesting is that you will see commits that you did not create. That is because you ***forked*** a pre-existing repository. This means that you also get all of the commits previously committed on the repository. Yours may look different than the picture above since I used a test repository when writing this manual.

Pushing your changes

When we ran the `git commit` command above, we committed our changes to our *local repository*. Remember when we cloned the repository using the `git clone` command? Well, we also need to ***push*** our changes back to the *remote* server (also known as the Gitlab instance).

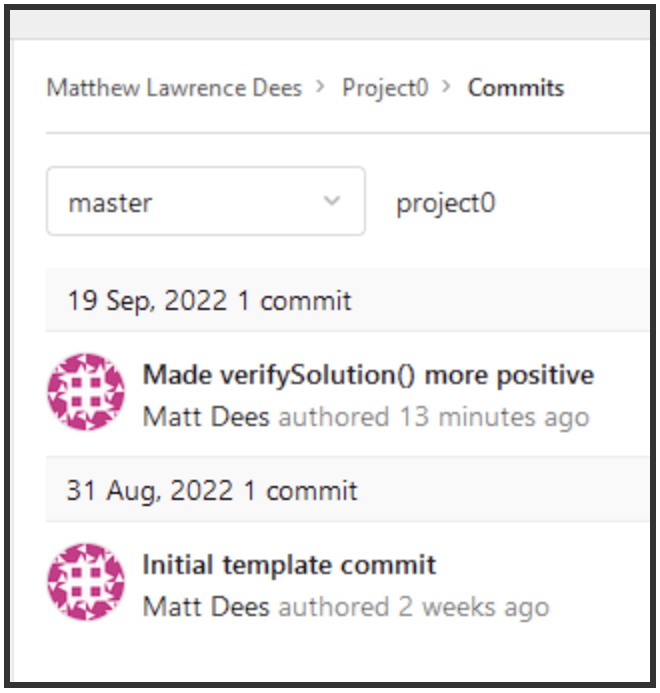
THIS STEP IS VERY IMPORTANT. IF YOU DO NOT PUSH YOUR CHANGES WE CANNOT GRADE THEM.

You can do this by running `git push`

```
ics46@ics46-shindler-2022spring : ~/project0 $ git push
Username for 'https://gitlab-ics46-test.ics.uci.edu': mldees
Password for 'https://mldees@gitlab-ics46-test.ics.uci.edu':
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 415 bytes | 415.00 KiB/s, done.
Total 4 (delta 2), reused 0 (delta 0)
To https://gitlab-ics46-test.ics.uci.edu/mldees/project0.git
   3b31137..f3c613e  master -> master
```

Pretty easy right?

You can verify that your changes have been pushed by navigating to the Gitlab instance (https://gitlab-ics46-f22.ics.uci.edu/<UCI ICS ID>/project0/) and clicking the **Commits** button.



We can now see that my commit has been pushed to the remote server and will be collected by the grading script.

TLDR.

1. Fork the project template here https://gitlab-ics46-f22.ics.uci.edu/mikes/project0
2. Clone your copy of the project repo:

   `git clone https://gitlab-ics46-f22.ics.uci.edu/<UCInetID HERE>/project0.git`

3. Modify source files with code changes
4. Stage your changes using git:

   `git status` (to see which files you modified)
   `git add .` (to add all changed files, instead of specifying all files in the current directory with period, you can choose specific files directly)

5. Commit your changes to your local repository

   `git commit -m "COMMENT"` (create a commit composed of the added files you specified in step 3 with a useful comment).

6. Push your ***commits*** to the remote server so they are backed up and can be graded.

   ```
   git push
   ```

The Project Template

A typical programming assignment will be provided with a few scripts.

- `build` : If you run the *build* script, it will (appropriately) build your program. If you want to build a program that runs through a main that you created, you will need to run “build app.”
- `run` : this will run the program that was just built. The default is that it will run the unit testing of what you just built; if you want to run the program that begins in app/main.cpp, run this as `run app`.
  To check your code for memory leaks, use the `--memcheck` flag when running your built program.
- `clean` : this will get rid of artifacts related to building the program in the past. This is useful if you want to recompile the entire program from scratch, for example.

Whatever C++ code you write should be placed into the app directory inside your project directory. When you're reading to compile it, change into the project directory (like `~/project0`) and issue the command `./build app`. (Note the dot and the slash in front of the word "build", and the space between "build" and "app"; those are important.) If compiling is successful, then issue the command `./run` to run your program.

If you want to run the unit tests, which your professor advises as a great way to test programs, use the command `./build gtest` and `./run gtest`. This will run a series of unit tests. Some starting point tests are provided for you in directory `gtest`, file `tests1.cpp`. These are a starting point, and are not intended as comprehensive tests. While getting less than 100% of these to pass will guarantee you do not get 100% on the project, passing each of these does not make any similar guarantees for the project itself. You are encouraged to test your code comprehensively. For the portion requiring templating, be sure to create cases that involve non-numeric types, such as `std::string`.

A good idea is to, after reading the requirements, think about what test cases your code would need to pass for you to be satisfied that you have written a good project. Then, add these test cases to tests1.cpp -- yes, even before you have written code to make them pass! The starting point code should help you see the format that a Google Test case expects, and the professor would be happy to help you if you are unable to do this on your own (make a private post on Ed with your desired test case(s) described and we’ll go from there). That way, when you are writing your program, you can check at any point if these cases have yet passed. In fact, some students write their test cases first, and then write their code to just pass the next test case that

is not yet passing, without “breaking” any previously passing ones. Then they backup their work (such as to a private git repository) and then continue. This is known as test driven development and is a very good software engineering practice.

Unit Tests

The /gtest folder in each assignment contains a file with some unit tests. You can run these after building your project and running with the `gtest` option. *The unit tests provided at the start of the project are not comprehensive, nor are they intended to be*. Failing any of these unit tests should indicate that you still have work to do on your project. However, successfully completing all of these *does not* mean that your program is fully correct. It is *your responsibility* to comprehensively test your code.

One recommendation for how to do this is to plan your testing before you write code. Write additional unit tests for each distinct piece of functionality you know you will need to have in your resulting program. While it is true that the starting point for code will not pass these, that is fine. You can then work on one unit test at a time, changing your code enough to implement the functionality that will cause it to pass without affecting the other unit tests. This also has the advantage of breaking a large project into smaller tasks, which should make it feel more manageable for you. It also means that if you have an hour between classes, there is a manageable part of your project you can work on, and measure the success of that hour. This technique is known as *test driven development*.

Project Submission

At the time the project is due, we will collect, from your repository, the most recently committed and pushed version of your project on the **main** branch. This is the “default” that we will grade. The good news is that if you have been committing and pushing your code as you make incremental progress, then even if you forget to finish the project by the deadline, you won’t have a zero as you would under many other collection mechanisms.

It is possible that this is not what you want us to grade. For these situations, you will need to first push and commit what you want collected -- it is possible you already did this; for example, you might want something other than the *most recent* of your in-progress submissions graded. You will then need to fill out a form we provide, in which you will tell us which SHA corresponds to the push you want graded. The submission time for such is *when you submit the form*, regardless of the date on the commit being graded.

Grading Environment

A copy of the ICS 46 virtual machine will be made available to you. While you are not required to use it to develop your code, we will be using it for grading assignments. We will not entertain regrade requests that ask us to evaluate student code on another machine, nor any that ask us to grade anything other than what was submitted. You are strongly encouraged to comprehensively test your code on the virtual machine before considering your submission to be final.

Programming assignments that do not compile in our environment will earn zero points, even if minor changes to the code would have caused a much higher grade. **This has been a problem for some students in past quarters. Be very careful and be sure to test every function in your code, especially with any assignment that requires the use of generic programming (templates).**

You may use any standard function that you could implement trivially (e.g., std::min, std::abs) unless it is explicitly disallowed or if it would clearly solve a problem you are expected to solve in this assignment. If you aren't sure, feel free to ask.

For various reasons, students are encouraged to backup the code they are writing as they progress. When using the git ecosystem this means committing and pushing your changes frequently.

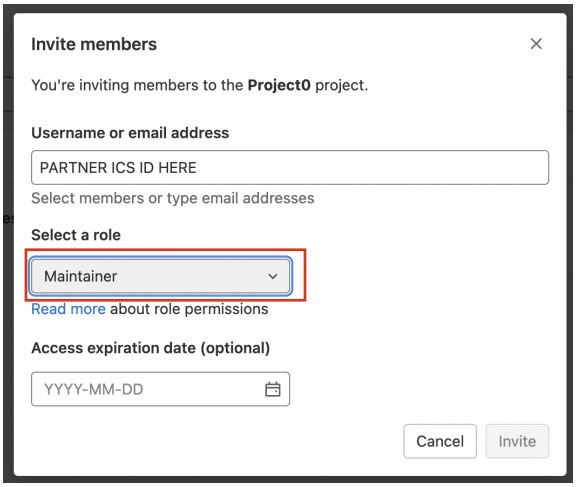
Partnerships

Some programming assignments will permit you to work with *one* partner. It is **not** required that you do so. If you do so, I expect you to work via pair programming. That is, you may **not** split the assignment, such as by having one person implement the data structure while the other person implements the function that uses the structure to solve a problem. I reserve the right to ask one or both project partners about the implementation and adjust the score for the group accordingly if I believe the work was split or one partner does not understand what was submitted. Similarly, any academic dishonesty arising from a group will be treated as an offense by both partners. *In past quarters, there have been situations where one partner plagiarized code without the knowledge of the other partner and both were held liable for the infraction. I believe this usually happens when a partnership splits a project.*

For any project that permits a partner, you will need to submit a form that will be provided in the project writeup. That form is often due a few days after the project *has been posted*. Late submissions for the form will not be accepted. Both partners will need to complete the form, which will require them to write their own UCINetID and also that of their partner. Partnerships where one partner does not submit the form on time, or where one or both partners submit the form incorrectly, will not be recognized. Your UCI Net ID is not your id number, nor is it your email address, although it is probably a substring of your email address. If you do not know what your UCI Net ID is, please read this link: UCInetID — Office of Information Technology

Partnerships *do not*, by default, carry over from one project to the next. You will need to submit the partnership form each time.

On the partnership form you will specify the UCI ICS ID of the repository you want graded. The grade for this repository will be assigned to both partners. The owner of said repository will need to add the other partner as a *maintainer* of the repository in order to contribute. This is done by clicking **Project Information > Members > Invite Members.**



Compliance With Project Requirements

We reserve the right to review your code to ensure compliance with project directions; failure to comply with project directions may result in a decreased score. This may reduce your score to zero. If we believe you were attempting to deceive us with code that will pass automated test cases without complying with project restrictions, we may consider it academic dishonesty, which will result in a report to AISC and an F in the course. It is possible that we will return a grade to you before we detect this; a grade reported to you does *not* mean we have agreed that your project is in compliance. I expect this policy to apply to very few, if any, students. An example of non-compliance with project requirements would be if you are asked to implement a linked-list based queue and your code does so via a `std::vector`. Such code would likely pass many automated test cases but would not be in compliance with project requirements, and would only be detected by code review. In order to expedite grades for the vast majority of students who are going to comply, we will likely return grades before inspecting code manually.

Late Policy

Every programming assignment is due at **7:30 AM Irvine time** on the day listed. Late submissions are accepted for all *except* project 0. Every hour, or fraction thereof, an assignment is late being turned in reduces the grade by 1% of what the submission would have

been worth had the student done so on time. There is a small and undisclosed grace period early in each hour that is considered part of the previous hour for purposes of submission penalties. Assignments two minutes late are probably not penalized, those fifteen minutes late almost certainly will be. Remember Murphy's Law and plan to submit your work on time!

Project 0 must be turned in on time and no late submissions will be accepted for it.

The largest late penalty, in terms of effect on your final grade in the class, will be forgiven. Remember that this only applies to programming assignments and that there is a **maximum** of 99 hours late, even for the project that will be your largest penalty. Failure to submit a programming assignment by the end of the grace period is not considered to be a "late" assignment for purposes of this policy.

If there are extenuating circumstances related to your ability to submit one or more assignments on time, please see your instructor to discuss how to handle this. The sooner you contact me, the more I can do to help you.

Grading

Your "raw score" is based solely on correctness : we will run some number of test cases for your code and, based on how many and which ones you get correct, you will earn some number of points. Each test case is worth some number of points and is graded based on whether or not your code correctly handles the test case. If it is determined that your program does not make an attempt to solve the problem at hand, you will not get these points, regardless of the result from testing. The tests will look a lot like the tests in your Google Test starting directory for this assignment; if you pass those, you're off to a good start, but it's not a guarantee. **You should always write more test cases than are provided to you at the start and be sure to comprehensively test your code. The provided test cases are to get you started.**

You will get a report on GradeScope with both your grade and a description of the test cases you missed, if any. We will not provide the test cases missed to you. If you have any questions about your grade, including a request for reconsideration, use GradeScope's regrade mechanism for that. See the syllabus for the grade reconsideration policy.

If we review your code and find your style to be sufficiently bad, we reserve the right to deduct points based on this, proportional to how bad the style is. If we do so, we will alert you to both the penalty and the reason. Here are some guidelines to follow when submitting your code:

- Include a reasonable level of comments. Do not comment every line, but do not omit comments either. A decent guideline is that if you were asked to explain your code six months from now, your comments should guide you to be able to do so.

- Use meaningful variable names where appropriate. Loop counters need not have a long name, but if you declare a `std::unordered_set`, give it a name reflecting what is in the set, not "mySet."

- Use proper scoping for variables. Avoid global variables except in rare circumstances. Pass parameters appropriately.

- Indent appropriately. While C++ does not have Python's indentation requirements for writing usable code, the guidelines of *readable* code are an issue here.

- Avoid vulgarity in your code and comments. Variable names, output statements, and the like should not make reference to topics that are not discussed in polite company.

- Remove debug output as appropriate; rather than commenting it out, delete it if it is unnecessary. If you think you might want to return to those debug statements, enclose them instead. Write something like this as a global variable (this is a case where such is acceptable):

  ```
  constexpr bool DEBUG_OUT = true;
  ```

  Then, when you need a debug statement, enclose it:

  ```
  if( DEBUG_OUT )
  {
        std::cerr << "Set has been successfully declared!" << std::endl;
  }
  ```

  When you want to remove the debug output, you can change the declaration to set the variable to false. You may wish to have multiple variables to control debug in various functions, either globally or having them local to functions. In the past, there have been several instances of students who have otherwise correct code but retained debug output statements -- and failed cases because they went over the time limit. An output statement takes more time than many instructions; I encourage you to set your program to run without output when it is submitted. We will not consider regrade requests that ask us to remove output then run the timed-out cases again.

  Similarly, if you keep your debug output lines in the file you turn in, having meaningful output statements is better than "code is here!" or "aaaaa." This is also true if you are going to ask someone (like the professor, TA, lab tutors, learning assistants) for help with debugging.

Alternate Environments

You are certainly within your rights to use something other than the ICS 46 VM to do your work this quarter, but you should be aware that you are bearing some risks by doing so:

- You'll be responsible for setting up and configuring the necessary software. As a rule, we won't be available to help configure environments other than the ICS 46 VM; you'll be on your own. For example, the provided project templates are specifically set up to work on the ICS 46 VM; making them work on something else will be up to you, and you might find this to be a tricky task, even if you believe you have all the same software we do.
- We'll be using the ICS 46 VM to grade your work, so if you write a program that compiles and runs on your own setup, but does not compile and run in the ICS 46 VM, you'll be the one to bear that risk. (This is not at all unrealistic, given differences in C++ compiler versions and the operating systems they run on, so don't discount this risk lightly.)
- When it comes time to submit your work, we expect everyone to use the provided gather script to gather their files for submission, so that everyone's submission is in the same format. We have test automation tools that depend on that. If you work outside of the ICS 46 VM, you'll have the additional problem of making sure that your submission is in the right format — and, if it's not, risk us not being able to grade your work, or that we refuse to grade your submission because we would have to perform special workarounds to be able to grade it.

At minimum, my suggestion is to get the ICS 46 VM set up, so that you can test your projects in it before submitting them and then run the gather script before submission, even if you prefer to do your day-to-day work elsewhere.