

Lab 0: Repository Setup & File Interface

[New Attempt](#)






Due Jan 16 by 11:59pm **Points** 100 **Submitting** a text entry box **Available** after Jan 9 at 12am

Lab Overview

In this course, you will be working individually to build several essential components of an RDBMS from scratch with the TacoDB framework (written in C++11). We will have four major labs throughout the semester and a warmup lab 0 (this one). Each lab consists of several labs that correspond to independent tasks. You will need to set up your **private** project repo through Github classroom. We will have scripts to pull your code for testing and grading automatically. Please follow the submission deadline strictly so that your code can be graded correctly.

The goal of lab 0 is to help you set up a development environment, catch up with the basic C++ programming skills on a POSIX-compliant environment, and finish the first component of the database storage module -- the file interface. At the end of this lab, you should be familiar with Git and Github; be familiar with the project codebase and its build and test system, and understand the semantics of the Linux I/O syscalls.

Repository Setup

You should also get familiar with basic shell and git operations, basic Github workflow, and C++11 if not already. Here're a few good resources: [POSIX shell](https://www.grymoire.com/Unix/Sh.html) , [git](https://docs.github.com/en/get-started/getting-started-with-git) , [Github](https://docs.github.com/en/get-started/quickstart) , C++11: [here](https://isocpp.org/wiki/faq/cpp11) , and [here](https://en.cppreference.com/w/cpp/11) .

System requirements: You'll need a desktop or laptop with an x86-64 CPU and a reasonably recent Linux distribution installed. The code is tested on Ubuntu 20.04 (on a bare-metal machine, virtual machine, WSL, or WSL2) and Fedora 35. It may or may not compile on other systems, and even if it compiles, it may not function as intended (especially if you are using a laptop with ARM CPU such as Apple M1, Microsoft SQ1 or SQ2).

The following is a list of software packages you need to install:


- git
- CMake (version ≥ 3.13)
- pkg-config
- C11 and C++11 compliant compilers, e.g.,
 - gcc and g++ (version ≥ 6 , recommended)
 - or clang and clang++ (≥ 3.3 , not tested)
- make
- autoconf
- bash as well as grep, sed, awk, etc.
- python3

The Autotrader and our final test server are both installed with Ubuntu 20.04 and gcc/g++ 9.3.

Step 1: Create your submission repository

Accept the Github Classroom invitation at <https://classroom.github.com/a/lp1hq099>  (<https://classroom.github.com/a/lp1hq099>).

This will create a private GitHub repository for you in which you will store your code. The repository is initially empty.

Step 2: Download the lab 0 handout file [here \(https://psu.instructure.com/courses/2240486/files/144603328?wrap=1\)](https://psu.instructure.com/courses/2240486/files/144603328?wrap=1). 
(https://psu.instructure.com/courses/2240486/files/144603328/download?download_frd=1) , and put it in a place where you'd like to set up your local Git repository. Then enter the following commands. You can find <repo-url> in the home page of the repo you created in the last step. Click the green button on the home page and select SSH option. The repo-url should look like `git@github.com:PSU-CSE541-SP23/cse-541-project-<psuid>.git`

```
# change into the directory where you put lab0.tar.gz
cd <dir-where-you-put-lab0.tar>
# extract the tarball
tar xf lab0.tar
# change directory
cd lab0
# setting up the git repo
./setup_repo.sh <repo-url>
```

If everything goes well, the script will print `Repo setup is finished. Here are a few post-setup steps to follow:...` Please follow the post-setup steps.

Step 3: Build the code. We use `cmake` as the build system. Most likely, you will not need to modify any of the `CMakeLists.txt`, but you should know how to invoke the `cmake` [↗\(https://cmake.org/cmake/help/latest/manual/cmake.1.html\)](https://cmake.org/cmake/help/latest/manual/cmake.1.html).

Here's how to create a debug build in the `build` directory:

```
cd <dir-to-local-repository>
cmake -B build .
cd build
make
```

The first build will also build a few dependencies in `external/` directory: [Abseil](https://abseil.io/) [↗\(https://abseil.io/\)](https://abseil.io/), [GoogleTest](http://google.github.io/googletest/) [↗\(http://google.github.io/googletest/\)](http://google.github.io/googletest/), [Jemalloc](http://jemalloc.net/) [↗\(http://jemalloc.net/\)](http://jemalloc.net/). Future builds will be faster as long as you do not remove the `external/` directory locally. You don't have to run `cmake` again in future builds either -- just run `make` in the `build/` directory.

Step 4: Run the tests locally with `ctest` [↗\(https://cmake.org/cmake/help/latest/manual/ctest.1.html\)](https://cmake.org/cmake/help/latest/manual/ctest.1.html) (which comes with cmake):

```
cd build
ctest -V
```

The code should be successfully built at the moment but none of the new tests should pass. We recommend that you make a new branch for each lab and/or create a tag for the starting point, in case you need to revert back to an earlier point.

Tasks

Use the Linux I/O syscalls (e.g., `open(2)`, `close(2)`, `pread(2)`, `pwrite(2)`, etc.) to implement the `FSFile` class in `src/storage/FSFile.cpp` and `src/storage/FSFile.h`. See the header and source file for requirements and hints.

Note: you may ignore all thread-safety-related requirements since we will be building a single-threaded database.

Testing and debugging: For lab 0, there are 14 basic tests `BasicTestFSFile.XXX` defined in `tests/storage/BasicTestFSFile.cpp` and another 14 basic tests `BasicTestFSFile.XXXNoFallocate` with an additional `--test_never_call_fallocate` argument. The `NoFallocate` variants test the fallback implementation of `FSFile::Allocate()` when `fallocate(2)` is not supported by your OS or file system. Your implementation should pass each of these 28 tests within tens of milliseconds. There are also 2 hidden system

tests `SystemTestFSFile.TestAllocateLarge` and `SystemTestFSFile.TestAllocateLargeNoFallocate`, which are not visible to you. In later labs, the system tests are those that are more complex, run on larger data, and/or have a longer running time. Nevertheless, they should not fail as long as your implementation satisfies the lab requirements. For that reason, system tests may only be tested on our end or in certain cases, we will run a final test on a dedicated test server after the lab deadline to determine the grade.

You may use `ctest -V` to run all the tests. Alternatively, you may test and/or debug an individual test by invoking its binary directly. For instance, the following shows how to test `BasicTestFSFile.TestCreateFile`:

```
cd build
# to test BasicTestFSFile.TestCreateFile
./tests/storage/BasicTestFSFile --gtest_filter=BasicTestFSFile.TestCreateFile
# to list more options
./tests/storage/BasicTestFSFile --help
```

You may debug your code using GDB and/or log messages. We provide a `LOG()` macro that you can use to print any messages in your code (see `include/base/logging.h` for details). As a concrete example, you may use the following code to print the value of a variable `x`:

```
int x = 0;
// do something with x
LOG(kInfo, "the value of x is %d", x);
```

Note that the log messages are all disabled by default in the tests because we do not want to see too many log messages when we are testing the functions with errors. To enable the log messages, run the test with an additional `--disable_logs=false` argument, e.g.,

```
cd build
# run BasicTestFSFile.TestCreateFile with log enabled
./tests/storage/BasicTestFSFile --disable_logs=false --gtest_filter=BasicTestFSFile.TestCreateFile
```

Submission Guideline

When you are ready to submit the lab, push your code to Github and find the latest commit hash. Git commit hash can be found on the Github website or through the command ``git log``. Copy and paste the commit hash into the text box for this assignment.