Added README.md.
Gene Stark authored 4 months ago

34baeb47

README.md 29.8 KB

CSE320 Spring 2021

In this course you will be using Linux as your primary development environment. In addition, we will be providing you with a git repository hosted on a department GitLab server. This document will briefly explain the course tools and outline the required setup for this course.

Setting up your CSE320 Git repository

Git is an open-source distributed version control system. We will use git repositories to manage your homework submissions. In addition, the use of git allows the Professor and TAs to access and view your your code remotely in order to assist you. While some students may be familiar with version control and git, we ask that everyone complete the following tutorial and instructions. This will ensure that everyone in the course has the same background knowledge and can submit their homeworks.

We are using a CSE department supported git web interface, called gitlab. This is similar to github, bitbucket, etc. It is an interface to help manage git repositories. These services are INTERFACES to git, not git itself. You *may not* use external repositories as we will use the repo provided to you to grade your submitted work and share gradesheets with you.

To setup your repository:

1. Navigate to https://gitlab02.cs.stonybrook.edu and log into it with your CS email account (user name only, do not include the `@cs.stonybrook.edu`). If you forgot your CS email password you can reset it by following the instructions here. If those instructions fail, please email `rt@cs.stonybrook.edu` requesting a password reset. A response may take up to 24-48 hours.
2. Once you have logged in the creation of your repo will be triggered. Normally this will occur within a few minutes. If not, then send an email to `cse320@cs.stonybrook.edu` and we will look into it. Sometimes the 'bot responsible for creating the repos has to be reset.

Setting up Linux Environment

Since C is a systems level language, frequently the behavior from one person's computer to another can vary. In the past, we have provided a common server for students to use, but this presented a few problems. When you wanted to compile your assignment, you would have to continuously transfer the file to the server and then compile it. If you had any mistakes, you would have to either edit it on the server or make the change locally and upload it again. This became very tedious which often led to students compiling and testing locally on their own machines. This was not always a good idea as something that seemed to work for you didn't always work for the grader which caused many issues. Also, many tools, which assist in locating and fixing errors in C code, do not exist in Windows and OSX environments. So students who installed operating systems such as Linux were at an advantage over the students who did not.

> 🤓 This document will also outline the homework management and submission process. In this class, you will be creating increasingly complex C projects which may involve many files. To satisfy these requirements, we will be using git to manage & submit your homework assignments.
>
> 🤓 While we will try to provide the basics for what needs to be done, it will ultimately be up to you to learn how to use these tools.

To help alleviate the above issues and to setup a local environment with the necessary course tools, you must install your working environment using one of these two options:

- Option 1: A Virtual Machine running Linux (Encouraged Option)
- Option 2: Multi-Boot/Install Linux on your machine

Option 1 is encouraged for the following reasons:

- Quick setup
- Ease of use in your native OS
- Easy to reset if errors in VM environment
- All course tools are pre-installed
- Simulate multiple cores on a single core system

We have put a lot of effort into setting up a pre-configured VM. If for some reason you are unable or unwilling to use this, we have provided basic instructions for Option 2 with a script to install all the course tools.

If you choose option 2, you should have some idea what you are doing, already be comfortable with Linux, and be aware that we probably won't have the resources to debug any issues you might encounter. If you deviate in any other way from these procedures, it is completely at your peril.

Option 1: A Virtual Machine running Linux

Students often use either VMware or VirtualBox to run virtual machines. We recommend that you use VirtualBox. It is free, and it runs on all of the most popular platforms.

In order to run a virtual machine, your machine must support 64-bit hardware virtualization. Most machines built after 2006 should support this. However, not all machines have the option enabled. You may need to modify your BIOS settings to enable this feature. As each machine has a different BIOS, it is up to you to find and enable this feature on your own machine.

Download and install the VirtualBox platform package appropriate for your computer from this site.

> ❗ Because of recent changes made to the way VirtualBox interfaces with the graphics drivers on various platforms, it is important that you make sure to install VirtualBox version 6.1 or greater. With older versions, the course VM image will probably not be able to access the display properly.

Running the Linux VM

We will be using Linux Mint 20 "Ulyana" -- Cinnamon as this semester's OS. We have taken the time to set up the VM so it simply needs to be opened in your virtualization program. The provided Linux virtual machine has all the tools required for various aspects of this course; for example, homework submission is pre-installed.

To get started, download the VM from here: Google Drive (it's over 4 gb so give it some time). This should result in your having a file called **CSE320_Spring21.ova**. This can be imported directly into VirtualBox by choosing "Import Appliance" from the "File" menu and then browsing to select the file you downloaded. Click "Next", review the VM settings, and then click on "Import". Once the import has completed, you should have a VM called "CSE 320". Select this and click on "Start" to boot the VM.

Login Info

Upon booting, you will be automatically logged in as user `student`. The login info for your reference is:

Username	Password
`student`	`cse320`

You will need the password in order to obtain superuser access via `sudo` to install software, and you might need to enter both the user name and the password if the screen lock should kick in after you have left the VM idle for some time.

VirtualBox Guest Additions

The VirtualBox Guest Additions are software components that are added to the guest operating system that runs in your VM, to make the VM more convenient to use. Examples of things in the Guest Additions are accelerated video drivers, support for clipboard and drag-and-drop between the VM and the host system, ability to resize the VM window, and so on. There is a version of the Guest Additions installed in the VM, but since the Guest Additions need to match the version of VirtualBox that you are using, you should reinstall them. To do this, you should start the VM, then from the "Devices" menu (probably in the titlebar of the VM window, or wherever top-level application menus appear on your system) select "Insert Guest Additions CD Image". This might cause a CD image to be downloaded over the network. If the system offers to auto-run the CD, allow it to do so. Otherwise you might have to use file manager (under Linux Mint) to open the CD manually. Once started, it can take several minutes for the installation to complete.

VM Snapshots

If you choose to install additional tools or other programs to your environment, you may want to take a snapshot of your VM. This may save you the time of installing your additional software again, in the unfortunate event of an unusable VM. Refer to the appropriate VirtualBox documentation to learn how to take a snapshot of your VM.

Option 2: Multi-Boot/Install Linux on your machine

> Remember, if you choose this option, you should have some idea what you are doing, already be comfortable with Linux, and be aware that we probably won't have the resources to debug any issues you might encounter. If you deviate in any other way from these procedures, it is completely at your peril.

Install Linux Mint 20 "Ulyana" - Cinnamon 64-bit or 20.04 Ubuntu variant (as long as you are using gcc 9.3.0) as a dual-boot or fresh install.

Clone the CSE320 course tools (https://gitlab02.cs.stonybrook.edu/cse320/course_tools) repository into your Linux environment. You may need to install git first.

Follow the README in the `course_tools` repo.

Working in Unix

We understand that many of the students taking this class are new to CLI (Command-line interface). You can find a quick crash course https://learnpythonthehardway.org/book/appendixa.html.

> 🤓 For more advanced usage refer here. This is a REALLY good resource so we recommend bookmarking it for later reference.
>
> 🤓 It is **very** important that you properly shut down the Linux Mint operating system when you are finished using it, rather than just "X-ing out" the VirtualBox VM window. The latter is equivalent to going and yanking your desktop PC's power plug out of the wall without shutting down Windows, and it can cause data loss and even corruption. Use the shutdown icon from the "Mint" menu in the lower left corner of the desktop to shutdown Linux Mint. At that point, it will be safe to power off the VM.
>
> 🤓 Depending on the host system on which you installed VirtualBox, "keyboard integration" and "mouse integration" might or might not be supported. If they are supported, then you will be able to fairly seamlessly move your mouse in and out of the VM window and what you type on the keyboard will go to the proper place. If these features are not supported, then you will need to click on the VM window in order to use it, at which point the mouse and keyboard will be "captured" by the VM. In order to regain control of the mouse and cursor, you will need to press the "host key", which is identified at the right-hand side of the bottom icon tray of the VirtualBox window. On some systems, the default host key is "Right Ctrl".
>
> 🤓 To open a terminal window, you can click on the terminal icon (which should be fairly evident), or you can press CTRL + ALT + T.

Text Editor

A *good* basic text editor is the key for C development.

We have pre-installed Sublime Text with plugins such as a C linter to assist with C development on the given VM. A linter displays compiler errors on top of your code much like an IDE. If you do install another editor we recommend looking into a similar feature described as it will aid development.

You may use another text editor if you so desire. Some popular ones are Atom, Vim, Emacs and VSCode. Each have their own linters that you can look into installing.

DO NOT install and use a full IDE (Clion, Netbeans, or Eclipse); there are many parts of the compilation process that are hidden from you. Not only would you miss out on valuable information pertinent to the course but your project is not guaranteed to build in an environment separate from the IDE.

Homework Management & Submission

Setting up your CSE320 repository

Once your repository has been created on gitlab, you must clone it in your Linux environment. Open a new terminal window and type `git clone GIT_URL` . You should replace `GIT_URL` with the URL to your repository. You can find it by navigating to your projects page on GitLab and selecting the https option.

> Your repo should be cloned into your home directory (`/home/student/` or AKA `~/`)

Alternatively if you add an ssh-key to your gitlab account you can clone, pull, push, etc. using the URL under the SSH option (**highly recommended** An SSH key can be done at any time).

References: - Generating SSH key - Adding SSH key to Gitlab

First Commit to your Repo

Open a terminal and from the home directory enter the following command: (replacing REPO_NAME with your repo's name)

```
$ subl REPO_NAME
```

The text editor, Sublime, will open and your repo's contents will be shown on the sidebar. Open the `README.md` file and add the text with the following information relevant to you.

```
# FIRST_NAME LAST_NAME
## ID_NUMBER
:FAVORITE_EMOJI:
PROFESSOR_NAME - SECTION_NUMBER
```

You can find your favorite emoji code among these https://gist.github.com/rxaviers/7360908. After that you can save and close the file and return to your terminal.

In your terminal, type the following commands, replacing `EMAIL` with your CS email address and `NAME` with your name:

```
$ git config --global user.email "EMAIL"
$ git config --global user.name "FIRST_NAME LAST_NAME"
$ git config --global push.default simple
```

NOTE: This will change your settings for all repos. If you want to have different settings for other repos on your machine then omit `--global`

Change directories into your repo `cd REPO_NAME`

Then run the following commands:

```
$ git status
$ git add README.md
$ git commit -m "My First Commit"
$ git push
```

> The `git push` command will prompt for username and password if you used HTTPS.

The output will look **similar** to:

```
$ 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 checkout -- file..." to discard changes in working directory)

    modified:   README.md

no changes added to commit (use "git add" and/or "git commit -a")
$ git add README.md
$ git commit -m "My First Commit"
[master XXXXXXX] My First Commit
 1 files changed, X insertions(+), X deletions(-)
$ git push
Counting objects: 4, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 980 bytes | 0 bytes/s, done.
Total 4 (delta 2), reused 0 (delta 0)
To ssh://git@gitlab02.cs.stonybrook.edu:130/CSE320_Fall20/REPONAME.git
   XXXXXXX..XXXXXXX  master -> master
Branch master set up to track remote branch master from origin.
$
```

This is the basic usage of git. We check the `status` of which files are tracked/untracked. Then we `add` them and we `commit` them along with a message. Lastly and most importantly we `push` them to the remote repository on the gitlab server. If the push was successful, you can navigate back to the page `https://gitlab02.cs.stonybrook.edu` and select your repository. Inside your repository, select the files option on the left menu. You should now see the file `README.md` with the contents you added to it.

> 😱 Once a commit has been made, its contents cannot be changed. In addition, the GitLab server has been configured so that it is not possible to delete any commits that have been pushed to the "master" branch. This means that any junk you commit to the master branch and push to the server will persist there forever in your repo, as well as in copies that we have to store. In view of this, it is is important that you take great care not to commit junk files, especially files that are very large or binary files that are generated by the compiler. Each time you commit, you should first use `git status` to carefully review the set of files to be committed. Use `git reset` to remove any files that are staged for commit but should not be. We strongly recommend that you *never* use commands such as `git add .` or `git add --all`, as these have the potential to add a lot of junk to your commit. Instead, `git add` each file individually, after perhaps using `git diff` to remind yourself of the reason for the commit and to see if the changes are as they should be.

Git Tutorial

We recommend you complete Codecademy's git tutorial found here if you are unfamilar with git.

If you're interested in learning more information about git or expanding your knowledge, refer to these references:

- git-book - Chapter 2 is a MUST read chapter, checkout git aliases!
- Learn Git Branching - An interactive tutorial on git branching
- git cheat sheet

Homework 0

Obtaining Assignment Code

1. Navigate to your repository directory (`cd ~/REPO_NAME`) in your VM (using the terminal).
2. An assignment, such as this one, will tell you the code is located at a particular address. For `hw0` it is: `https://gitlab02.cs.stonybrook.edu/cse320/hw0.git`
3. Add this remote repository as an additional remote into your existing repository. We will name the new remote HW0_CODE.

   If you use HTTPS:

   ```
   $ git remote add HW0_CODE https://gitlab02.cs.stonybrook.edu/cse320/hw0.git
   ```

   If you use SSH:

   ```
   $ git remote add HW0_CODE ssh://git@gitlab02.cs.stonybrook.edu:130/cse320/hw0.git
   ```

4. Fetch all the refs in this new repository. This command will prompt for username and password if you used HTTPS.

   ```
   $ git fetch HW0_CODE
   ```

5. Finally, merge and commit the files from the `HW0_CODE` remote's `master` branch into your existing repository's `master` branch.

   ```
   $ git merge -m "Merging HW0_CODE" HW0_CODE/master
   ```

   > 🤓 If you get an error mentioning 'unrelated histories' try again adding this flag: `--allow-unrelated-histories`

6. If you type the command `ls` you should now see a directory called `hw0` .
7. Push these base files to your remote repository (gitlab). This command will prompt for username and password if you used HTTPS.

   ```
   $ git push
   ```

Your Homework 0 Working Directory

The directory structure of your repo will now look **similar** to this. Use `ls -a` or `tree -a` to see the hidden files that begin with `.`

```
YOUR_REPO
├── .git
│   ├── ...
├── .gitignore
├── .gitlab-ci.yml
├── hw0
│   ├── academic_honesty.txt
│   ├── include
│   │   └── hi.h
│   ├── Makefile
│   ├── README.md
│   ├── src
│   │   ├── hi.c
│   │   └── main.c
│   └── tests
│       └── test.c
└── README.md
```

Information about each file is explained below.

> 🤓 Enter `subl REPO_NAME` (or `subl .` if you are in your repo already) as you did before to easily follow along and look inside each file

- `.gitignore` - This is a file that tells git to ignore certain directories or files that you don't want committed. For example, the `bin` and `build` directories are ignored. This is because we don't want executables and other generated binary files pushed to your remote repository, only source code.
- `.gitlab-ci.yml` This is gitlab's own continuous integration configuration file, explained in a later section.
- `hw0/` - This is your first homework directory, throughout the semester we'll be adding each homework directory in this fashion 'hw#' where # is the homework number. Inside the `hw0/` directory, you will find:
  - `README.md` - This is a file where you can detail notes about the project.
  - `Makefile` - This is your ultimate compilation automation tool. The program `make` will use this file to properly compile your assignment.

- `include/` - This is where we keep our `.h` headers. Unlike Java, C is a one pass compilation language, which keeps the overhead of creating symbol tables and structures low. Since all functions must be defined before use, we utilize a header file to make the symbols available across multiple files.
  - `hi.h` - This is our header file for `hw0`. **Examine the contents of this file.**
- `src/` - This is where we keep our `.c` source files. These files contain the actual definitions of the functions declared in our headers.
  - `main.c` - This file contains the C main function, in this course you will **ALWAYS** need to keep your main function in its own C file isolated from the rest of your functions that you implement.
  - `hi.c` - The helper function, `hi`, is defined (implemented) here. Each function **does not** need its own file, only `main()` needs to be in its own file.
- `tests/` - This is where we keep our unit tests. There is an EXCELLENT unit testing framework called criterion that we will be using in the course.
  - `test.c` - This file contains the implementation of our testing framework. The Makefile will compile these files with all of the non- `main.c` files. This gives the testing framework access to your helper functions.

> **Do not** modify or alter `.gitlab-ci.yml`, the `Makefile` or the `README.md` for any assignment unless otherwise instructed.

Academic Honesty Statement

In this course we take Academic Honesty EXTREMELY seriously. Read the statement in `academic_honesty.txt` using your favorite text editor or using the following command (type `man cat` for more information on this tool).

From your repository's root directory type:

```
$ cat hw0/academic_honesty.txt
```

Next, we will append the Academic Honesty Statement into your repository's README along with the date and your "signature" confirming that you have read the statement and agree with the policy and commit it to your repo.

From your repository's root directory type the following commands into your terminal, filling in `YOUR_NAME` with the appropriate information in the second command.

> 🤓 The second "crazy" command is an example of redirection which can be done between programs on the command-line. We will learn more about redirection and how it works later in the semester.

```
$ cd hw0
$ cat academic_honesty.txt <(echo "$(date -u) - YOUR_NAME") >> ../README.md
$ git add --all
$ git commit -m "Academic Honesty statement"
$ git push
```

CI File

This semester we want to ensure that students don't get caught by silly mistakes or overlook anything while working on their assignments. We will use GitLab's *Continuous Integration* feature to minimize such incidents. It is an automated tool that will make sure your work compiles and passes basic tests.

There is a `.gitlab-ci.yml` file in the base code. This file is used to set up a clean vm, compile your code, run your unit tests, and lastly run your program on the gitlab server. When looking on gitlab you will notice a red ❌ or green ✔. Each represents the result of a 'CI pipeline'. Go to the Pipelines tab to view the results of your run. We will provide this file with each homework. The CI runs when gitlab "gets around" to doing it so don't be alarmed if you don't see your result right away. Also, note that sometimes a "Runner System Failure" might occur due to problems on the server. A failure reported by CI system does *not* mean that "your commit failed" in the sense that what you committed and pushed failed to make it to the gitlab server; it means that an error occurred while the system was trying to compile and run what you committed. You can always use the `Repository` tab of the gitlab web interface to see what commits you have pushed to the server. If a commit is shown there, then it is safely on the server.

Hello, World!

In the terminal, navigate into the `hw0` folder of your repository (ie. `~/REPO_NAME/hw0/`). Open the file `include/hi.h` and examine its contents, read the comments, and follow the directions in the files. These directions step you through the base files in the HW to familiarize you with basic files structure and the included code complements.

In the terminal, type `make` to build the `hw0` base code. This will compile your c code into executables.

```
$ make
mkdir -p bin build
gcc build/hi.o build/main.o -o bin/hi
gcc -Wall -Werror -std=gnu11 -g -DDEBUG -I include build/hi.o tests/test.c -lcriterion -o bin/hi_test
```

An executable named `hi` and `hi_test` will be created in the `bin` folder of the `hw0` directory. You can execute either program by typing `bin/hi` or `bin/hi_test` from the `hw0` directory into the terminal.

Running `bin/hi` will print "Hello, World!" before exiting. Running `bin/hi_test` will fail a unit test and print the warning `"Assertion failed: say_hi() function did not say 'Hi'"`.

```
$ bin/hi
Hello, World!
$ bin/hi_test
[----] tests/test.c:15: Assertion failed: say_hi() function did not say 'Hi'
[FAIL] CSE320_Suite::test_it_really_does_say_hi: (0.00s)
[====] Synthesis: Tested: 1 | Passing: 0 | Failing: 1 | Crashing: 0
```

To do this assignment, modify the `say_hi()` function in `src/hi.c` to satisfy the unit test (i.e. `return "Hi"`). This is will now make the program do what the unit test expects and as a result pass the unit test.

Rebuild the hw0 executables by typing in ‘make’ to your terminal. Run the program again to make sure it satisfies requirements

```
$ make
mkdir -p bin build
gcc -Wall -Werror -std=gnu11 -g -DDEBUG -I include -c -o build/hi.o src/hi.c
gcc build/hi.o build/main.o -o bin/hi
gcc -Wall -Werror -std=gnu11 -g -DDEBUG -I include build/hi.o tests/test.c -lcriterion -o bin/hi_test
$ bin/hi
Hi, World!
$ bin/hi_test
[====] Synthesis: Tested: 1 | Passing: 1 | Failing: 0 | Crashing: 0
```

To save your code changes, `add` them to the staging area of git. `Commit` the changes to a local commit on your VM. Then `push` the commits to the remote server (gitlab).

You can be sure that everything compiles correctly if at some point a check appears next to your repo on gitlab.

```
$ 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 checkout -- file..." to discard changes in working directory)

    modified:   src/hi.c

no changes added to commit (use "git add" and/or "git commit -a")
$ git add src/hi.c
$ git commit -m "Hi Fix"
[master XXXXXXX] updated hi.c
 1 file changed, 1 insertion(+), 1 deletion(-)
$ git push
Username for 'https://gitlab02.cs.stonybrook.edu': REPO_NAME
Password for 'https://REPO_NAME@gitlab02.cs.stonybrook.edu':
```

How to submit with `git submit`

This semester you will be submitting all assignments using our custom git command: `git submit`

The usage for `git submit` is:

```
$ git submit [-c COMMIT_HASH] TAG
```

The **TAG** is which assignment you are tagging, for example: `hw0` or `hw1` or `hw2`. This is the format for all tags (with a few exceptions) that we will use this semester `hw#` where `#` is the assignment number.

The `-c` flag is optional. **COMMIT** is the SHA of the commit you wish to submit. In case you wanted to submit a different commit than your most current one you would just provide the SHA for the commit to be submitted. You can view your commit SHAs using the following command:

> The SHA is the alphanumeric string before the first hyphen.

```
$ git log --pretty=format:"%h - %s - %ar"
```

With `git submit`, you may:

- Submit your assignment **only** from the master branch.
- The commits that you are submitting must be on the master branch or merged into master (commit hash is on master).
- You may use other branches if you wish, but you cannot use git-submit from these branches.

```
$ git submit hw0
```

This will submit your latest pushed commit for `hw0`. You can submit as many times as you wish prior to any deadline.

How to check your submission

It creates a special branch in your repo called 'submit'. The only reason it is special is that only instructors have permission to push to it, hence why you need a special tool to submit your assignment. So the submit tool tags the commit that you want to submit and merges that commit into the submit branch under an authorized user.

Also, you should see a submission commit on the 'submit' branch. The submission commit is a commit with a commit message formatted as " submission commit". This you can see by navigating to Repository > Commits and selecting the 'submit' branch from the dropdown.

If you see both of these things, you have successfully submitted the assignment. A successfully submitted homework assignment will have a tag for that particular homework which you can see if you go to your project in Gitlab and navigate to Repository > Tags.

These are the tools and the dev environment you will be working with for the rest of the semester. We have provided you with these tools to ease the protocols of this course and prevent mishaps from occurring.

So that you do not delay your ability to start on `hw1`, you should make every effort to complete the steps above by **February 12, 2021**, 11:59PM. After that date it will no longer be possible to `git submit hw0`, however you should still make the commits to your repository described above prior to submitting `hw1`. Although `hw0` will not be formally scored and weighted into the final grade calculations, you will not receive credit for any other assignments in this course unless the signed Academic Honesty statement has previously been committed to your repository and thereby incorporated into the submissions for those assignments.