CS256 — Exercise 1

January 6, 2016

Due: Monday, January 11, 2016 before midnight (50 points)

In this exercise we will make sure everyone has a C++ compiler and git installed properly so that they can be used for future assignments.

1 Installing a C++ compiler

- If you are using a distribution of Linux, you should install packages for either `clang++` (recommended) or `g++`.
- If you are using Windows, you have a few options:
  1. You can download the Community edition of Microsoft's Visual Studio for free from `https://www.visualstudio.com/en-us/products/visual-studio-community-vs.aspx`.
  2. You can install MinGW from `http://www.mingw.org/` to get a Unix-like experience.
  3. You can create a Linux virtual machine and do your compiling and testing there. I recommend using VirtualBox, which is available from `https://www.virtualbox.org/`. This is probably worth doing for the final project because some students had trouble last quarter using Windows.

     This will also help you get some experience using Linux without requiring you to replace your existing OS. If you would like to use a virtual machine, I have a virtual machine image of Xubuntu prepared already. Let me know if you want more instructions to try it out.
- If you are using a Mac, I believe you can install Xcode for free which should come with a C++ compiler. Once it is installed, try running `clang++` from a terminal to see if it exists.

2 Installing and configuring git

1. - If you are using a distribution of Linux, git should be available in your package manager's repositories. For example, on Debian-based distributions you can use `apt-get install git` as `root`.
   - If you are using Windows, you can get a Windows installer from the git website at `http://git-scm.com`. When installing, the defaults should be fine. After the installation, you should have access to a program called git bash that will give you a Unix-like terminal. For the rest of the section, use git bash to run the commands.
   - If you are using a Mac, there should also be an installer available at `http://git-scm.com`.
2. From a terminal or git bash, run the following commands to configure your user:

```
git config --global user.name "Your Name"
git config --global user.email "youremail@cpp.edu"
```

3. Optionally, you can change some of the other options. For example, you can change the default editor to `nano` with:

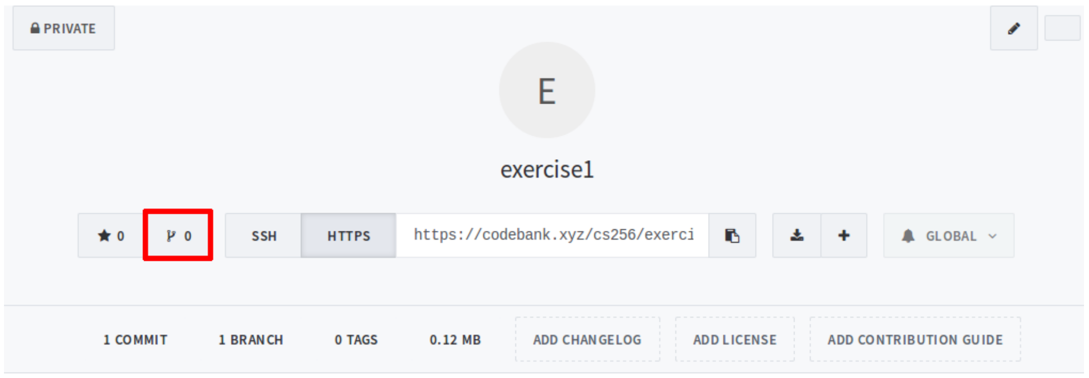
```
git config --global core.editor nano
```

   If you'd like to use a different difftool such as Meld, install it then set the following:

```
git config --global diff.tool meld
git config --global merge.tool meld
```

3 Project Setup

1. If you have not done so, register your account at `https://codebank.xyz` using your `@cpp.edu` email address and your bronconame as your username. After doing this, let me know so I can add you to the CS256 group.
2. Log in to `https://codebank.xyz`. You should see a project labeled `cs256/exercise1`.
3. Open that project and you should see a button as shown below:



   Press this button to "fork" the repository. This means that you will make a copy of this project under your own user instead of the `cs256` group.
4. Once you have completed forking the repository, if you go back to your home page (click the Gitlab logo on the top left) you should see two projects. From now on, you will use your own copy of `exercise1` instead of the `cs256` version.
5. On your local machine, open a terminal (or git bash) and navigate to where ever you plan to store CS256 related files.
6. Next, run the following command to clone the repository from the gitlab server to your local machine where `username` is your bronconame:

```
git clone https://codebank.xyz/username/exercise1.git
```

   You now have a local copy that you can modify then push changes back to `codebank.xyz`.

4 Modifying the Code

Right now, the code is a simple "Hello, World!" program. Add one new line after it prints "Hello, World!" that prints "Goodbye, World!".

4.1 Submitting the Changes

1. Once you have modified your code, compiled it, and tested it, run `git status` to verify that you see `Hello.cpp` has changes that have not been tracked. You may also have new files if you have compiled your program. You can either delete them or commit them as well.
2. Next, use `git add Hello.cpp` (and compiled files if you want) to add them to the staging area.
3. Use `git commit -m 'added goodbye message'` to make your first commit.
4. Use `git push origin master` to push your changes to `codebank.xyz`.
5. Go to `https://codebank.xyz` to verify that your changes have been sent to the server.

5 Making Another Commit

Next, we will add another line to our file. Add a line that prints out the size of a `long double`. After doing this, compile and test your code.

5.1 Submitting the Changes

1. Once you have modified your code, compiled it, and tested it, run `git status` to verify that you see `Hello.cpp` has changes that have not been tracked. You may also have new files if you have compiled your program. You can either delete this or commit them as well.
2. Next, use `git add Hello.cpp` (and compiled files if you want) to add them to the staging area.
3. Use `git commit -m 'added line to print size of long double'` to make your second commit.
4. Use `git push origin master` to push your changes to `codebank.xyz`.
5. Go to `https://codebank.xyz` to verify that your changes have been sent to the server.

6 Bonus: if you want to go back in time

Git is now tracking the repository in three states:

1. The original forked code.
2. The code after adding the goodbye message.
3. The code after adding the line to print the size of a `long double`.

If you want to go back to see what the whole project looked like for any of these commits, we can do it by doing a checkout (think of it like checking out an old book in a library).

To try this out, do the following:

1. Run `git log` to get a list of commits. Notice the long string of text after the word commit for each one. This is a SHA-1 hash that identifies this commit.
2. As an example, suppose I have a commit labeled:

   ```
   commit 5c520ea9a7bb2ff77db9e9e53564bf56644d9694
   ```

To checkout this commit, you need to provide *at least* 4 characters of the beginning of the hash and it must be unique among the hashes (so you might need more than four characters).

The checkout is then done with:

```
git checkout 5c52
```

3. After doing this, notice that if you open the code files they look like they did when we first started. We are now looking at our project as it existed in that commit.
4. To return to the most recent commit, we can use:

```
git checkout master
```