CS380 — Exercise 1

January 4, 2017

Due: Wednesday, January 11, 2017 before midnight (40 points)

In this exercise we will expand on the simple server and client programs from the first lecture and get some hands-on practice using `git`.

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. You can use any name or email, but git requires identification to work properly.

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

3. Optionally, you can change some of the other settings. 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
   ```

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.
2. Log in to `https://codebank.xyz`. There should be a project visible at `https://codebank.xyz/nmpantic/CS380-EX1`
3. Open that project and you should see a button labeled "Fork" 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 my user.

4. Once you have completed forking the repository, if you go back to your home page (click the Gitlab logo on the top) you should see your own copy of the project.
5. On your local machine, open a terminal (or git bash) and navigate to where ever you plan to store CS380 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/CS380-EX1.git`

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

Modifying the Code

Right now, the code is the same as the `TextServer.java` and `TextClient.java` code from the first lecture. You will now modify it to do something slightly different.

You will be implementing an echo server and client. It will work as follows: the client will connect to the server and the user can type in some messages. These messages are then sent to the server. The server should read in each message line-by-line and echo it back to the client.

Additionally, when a client connects to the server, the server should locally print the client's address. When a client disconnects, the server should print a message saying that the client disconnected.

On the client side, you should have a prompt display for the user to enter text with the label `Client>` and another prompt that shows the server's responses with the label `Server>` .

An example test run should look like this:

Server:

```
$ java EchoServer
... (time passes, this doesn't get printed)
Client connected: 127.0.0.1
... (time passes, this doesn't get printed)
Client disconnected: 127.0.0.1
```

Client:

```
$ java EchoClient
Client> Hi server
Server> Hi server
Client> Testing testing
Server> Testing testing
Client> exit
```

If the user enters `exit` the client program should close, disconnecting from the server.

Hint: Use a `BufferedReader` to read text on the server line-by-line and check the documentation for the `readLine()` method to see how to use it to determine when a client has disconnected.

Submitting the Changes

1. Once your programs work properly, run `git status` to verify that you see that the two files have changed.
2. Next, use `git add EchoServer.java EchoClient.java` to add them to the staging area.
3. Run a `git status` again to verify that they are ready to be committed.
4. Use `git commit -m 'EchoServer and EchoClient working.'` to make your first commit.
5. Use `git push origin master` to push your changes to `codebank.xyz`.
6. Go to the https://codebank.xyz website to verify that your changes have been sent to the server. The new commit should appear on the site in the "commits" section.

Multiple Simultaneous Clients

If you try to connect with a second client while one is already connected, you'll notice that the server does not respond. This is because the server program is currently handling the first client. If we want to be able to handle multiple clients, one option is to make our program multi-threaded.

We'll use a simple approach where we create a separate thread for each client that connects. To do this, you should modify what happens after the server program returns from the `accept()` method call. Wrap the code you have that handles communicating with one client in to a `Runnable`, pass it to a new `Thread` instance, and start that thread. Your server will then run that code simultaneously as it waits for a new client to connect.

Once you have your programs working for multiple simultaneous clients, we'll do another commit and push.

Submitting the Changes

1. Once your programs work properly, run `git status` to verify that you see that `EchoServer.java` has changed. The client should not have changed in this part.
2. Next, use `git add EchoServer.java` to add it to the staging area.
3. Run a `git status` again to verify that it is ready to be committed.
4. Use `git commit -m 'Implemented multiple simultaneous clients.'`.
5. Use `git push origin master` to push your changes to `codebank.xyz`.
6. If you run `git log` you should now see three commits listed.
7. Go to the https://codebank.xyz website to verify that your changes have been sent to the server.

Note: for full credit you must have both commits, one with the basic echo server and client working and one with the multi-threaded version. You will be graded based on your repository on the website.

Bonus: if you want to go back in time

This section is not graded and you don't need to submit anything. It is just here to show how `git` can be used for viewing code from old commits.

Git is now tracking the repository in three states:

1. The original forked code.
2. The functioning echo server and client.
3. The modified echo server that handles multiple clients.

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 my first commit is 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
   ```