Computer Science 241

Lab 1 (10 points)

Due Sunday, April 12, 2020 at 9:59 PM

This lab assumes that:

- You have successfully logged into a CS lab machine that is booted into linux. Please do this by connecting to a lab machine through `labs.cs.wwu.edu` via `ssh`. Instructions can be found here:
  `https://gitlab.cs.wwu.edu/cs-support/public/-/wikis/labs_ssh_balancer`.

- You know how to edit a text file using a command-line text editor, or you can run graphical applications remotely using X forwarding.

  The easiest command-line text editor to learn is `nano`. Tutorials abound, but this one seems nice:
  `https://www.howtogeek.com/howto/42980/the-beginners-guide-to-nano-the-linux-command`
  Other popular choices include `vim` and `emacs`. These have a significantly steeper learning curve, but they are extremely powerful once you do become proficient.

  X forwarding works out of the box if you're coming from linux; on MacOS, you need to install Xquartz, and on Windows you'll need to install VcXsrv. Please see the "Advanced Usage" section of the labs ssh load balancer instructions linked above for usage. If you get this working, you should be able to run graphical text editor such as `gedit` and see a window pop up on your screen. This program is running on the remote lab computer, but displaying on your screen - neat!

If you have trouble with any of this, please ask questions on Piazza, in office hours, or during one of the scheduled lab times on Thursday. We are here to help!

Overview

The purpose of this lab is to give you basic familiarity with some of the tools we'll be using in this class, namely **git** and **gradle**.

Git

Git is a *version control* system that is used to keep track of changes you make your code. We will use git in conjunction with Github, a web-based service that hosts git repositories.

A git repository is a place where your code lives. The internal data needed to track versions of your code is stored in a hidden folder (called `.git` at the base directory ("root") of your repository, but you don't need to worry about that - you'll use the `git` command-line tool to work with the version control system. Github hosts a *remote* copy of your code on their servers, which other people (e.g., me when I go to grade your work) can access and create separate copies of ("clone").

The basic workflow for using git and Github is to (1) make some changes to your code, (2) **commit** those changes to the repository to put them in the official record of version

history, and when you've reached a stopping point, (3) **push** to Github to update the hosted repository to reflect the changes you've committed to your local repository ("repo", for short). When you've followed this workflow and gradually made changes over time, git makes it easy to move forward and backward in history, for example to roll back to an old version of your code to find a version before a bug was introduced.

Gradle

Gradle is a *build system* that is used to take care of a lot of the logistics of building, running, and testing your code. It was created for Java, but it can be used for many languages. Once things are properly set up (we'll largely take care of this for you in this course), you compile your code with `javac` and run it with `java`, but instead use commands like `gradle build` and `gradle test` to compile, run, and execute tests on your code.

Keep in mind, this lab is just getting you started; you are not expected to become an expert in all the nuances of these tools yet—expertise will come with time and experience. The goal here is to get you ready to work on future labs and assignments.

Details

Complete the following steps.

0. Once you're logged into a lab computer, type "script" and press enter. This command begins recording a log of your commands, storing the log in a file called "typescript" by default. When you are done, the transcript will be part of your submission for this lab. If you need to stop working and resume later, you can type "exit" to stop recording. To resume recording, make sure you are in the directory containing the typescript file and run `script -a`; the -a flag tells the command to append your new log to the existing one, rather than starting fresh.

Setup

1. Assignments in this class will be done in GitHub repositories orchestrated by GitHub Classroom. Begin by logging into Canvas, finding the Lab 1 assignment, and clicking the GitHub Classroom invitation link found there. You will need to log into your GitHub account if you haven't already.

2. Once you have accepted the GitHub Classroom invitation, you will be given a link to your repository for this assignment. This link should be in the form of:

   `https://github.com/csci241-19w/lab-1-username`

   You should be able to click this link and see the repository on GitHub.

3. The git repository now exists on GitHub's servers, but you still need to **clone** a local copy of the code so you can work on it. You will need to choose a location for the local

lab1 repository and working copy. You may choose whatever location you like; here we will assume you choose `~/csci241/`. Note that the `~` indicates that the csci241 directory lives inside your *home directory* (usually /home/username, on linux systems), which is where you should be when you open a new terminal window. To create this directory, type `mkdir csci241`. Enter the directory using the cd command: `cd csci241`. You can list the contents of the directory (it should be empty) using `ls`.

4. Create a working copy of your code by cloning the repository from GitHub. You can copy the command from the green button that says "clone or download" on GitHub, or type it yourself:

```
# replace username with your github username
git clone https://github.com/csci241-19w/lab-1-username
```

   This should clone the repository into a new directory called `lab1-username`, which you should now see if you type `ls`.

   Basic Git Operations

5. Before you use Git commands for the first time, you need to tell it a little bit about yourself. You'll only need to do this once for each computer you use git on. Change directory into your freshly cloned repository and run the following commands, supplying your full name and email address:

```
# sub in your name
git config --global user.name "Your Name Goes Here"

# replace username w/ your wwu username
git config --global user.email username@wwu.edu
```

6. Now you will a writeup file and tell git to track it (that is, keep include it in your repository). Note that spelling, spacing and capitalization matter in the following commands:

```
touch writeup.txt
git add writeup.txt # run "git help add" for details
```

   This `git add` does two things: 1) it begins "tracking" `writeup.txt` and 2) it "stages" the changes to the file (namely its creation) so it will be incorporated in the next commit.

7. **Commit** your changes to the local repository:

```
git commit -m "Added empty writeup"
```

The text in quotes is the commit message; aim for it to be concise yet specific. Bad commit messages include `"Made some changes"`, `"stuff"`, and `"more edits"`. At this stage your changes have been stored on the local repository but not in the "remote" repository stored on Github. You can confirm this by browsing to the Github URL for your repository:

`https://github.com/wehrwein-teaching/lab-1-username`

You should see that writeup.txt is not listed among the files on GitHub.

8. To synchronize your local repo with the version on GitHub, push your changes from the local repository to the original one:

   `git push # sometimes it wants you to be more specific: git push origin master`

   Now refresh the Github URL, and you should see writeup.txt file.

9. Edit `writeup.txt`, so that it contains the line `1) First Last` where you replace `First Last` with your first and last names.

10. Stage these changes for commit:

    `git add writeup.txt`

    While `writeup.txt` is already tracked, this will stage the change.

11. Commit your changes:

    `git commit -m "Added part 1 (names) to writeup"`

12. `git status` lets you know the status of your working copy and local repository. Run the command and see what it reports. Now edit `writeup.txt` again, adding the line `2) Hobby: XYZ` where you replace `XYZ` with a hobby of yours. Check the status after making this edit. Check the status again after `git add`ing the writeup. Check the status again after committing, and then again after pushing.

13. Create another file in your repository called `username.txt`. Edit this file to contain exactly the following three items, separated by commas, with no extra spacing:

    - First and last name
    - WWU Username
    - Github username

    For example, Prof. Wehrwein's username.txt would read:

    `Scott Wehrwein,wehrwes,swehrwein`

Commit username.txt to your repository.

14. Learn your way around the following commands (use `git help` or search the web for details). Be sure to try out each command at least once and get familiar with how they work.

    (a) `git checkout` and `git reset` to undo a change to a file

    (b) `git rm` to schedule a file for deletion (first add a dummy file instead of deleting your writeup)

    (c) `git mv` to rename a file (if you move writeup, move it back after)

    (d) `git blame` to see who edited which lines when

    (e) `git log` to see your commit history

    (f) `git diff` to see unstaged changes to a local file

    (g) `git diff` to see changes between two different committed versions of `writeup.txt`

    Make sure that by the end of playing around with those commands your `writeup.txt` is back to being named `writeup.txt` and contains the two lines described above, and your `username.txt` also conforms to the specification given above.

Branching and Merging

Branching is a wonderful version control feature and one that git does quite well.

15. Read through Git's documentation on branching and merging:

    `https://git-scm.com/book/en/v2/Git-Branching-Basic-Branching-and-Merging`

16. Create a branch named `question3` and checkout that new branch. Edit your `writeup.txt` to add the line `3) Favorite song: ABC by XYZ` where you replace `ABC` with the track name of your favorite song and `XYZ` by the artist/band of the song. Add and commit your change to the branch, then merge the changes into the `master` branch. After the changes have been merged into `master`, delete your `question3` branch.

Java and Gradle

In this class, we'll be writing code in java. To facilitate the process of compiling, testing, and running code, we'll be using a build tool called Gradle. This replaces the command line workflow (using javac to compile and java to run) or the build-and-run functionality provided by IDEs.

Your repository already contains the directory structure used by Gradle to keep track of source code and compiled programs. You will find a file called Hello.java in the `src/main/java/lab1/` directory. This file contains a Java implementation of the canonical Hello, World program.

17. Compile the Hello World program by running the following command:

    ```
    gradle build
    ```

18. Whereas you may be accustomed to running programs directly using the `java` command, we will instead use Gradle to run the program for us. Run the Hello World program with the following command:

    ```
    gradle run
    ```

19. Edit `src/main/java/lab1/Hello.java`, modifying the program to print the contents of the 0th command-line argument instead of "world". Recall that the `args` parameter to the main method contains the command line arguments passed to a Java program.

20. Compile your modified code as above. To pass command-line arguments to your program via gradle, use the `--args` flag as follows:

    ```
    gradle run --args="your args here"
    ```

    A sample invocation of the modified program should look something like this:

    ```
    $ gradle run --args="241"

    > Task :run
    Hello, 241!

    BUILD SUCCESSFUL in 0s
    2 actionable tasks: 2 executed
    ```

21. Stage, commit, and push your changes to Hello.java:
    ```
    git add src/main/java/lab1/Hello.java
    git commit -m "Hello now prints command line arg"
    git push
    ```

Submission

At this point stop (by typing `exit` on the terminal) the `script` command you started at the beginning of this lab. This should write a transcript of your terminal session (including all of your git commands) in a file named `typescript`. Copy that file to your repo directory if it isn't already there, and then git add it, commit and push. Confirm that your `typescript` file is visible by checking that the file exists via the GitHub website.

Grading

This lab is worth 10 points, assigned as follows:

- 6 points: Basic git operations steps are completed and all listed commands have been executed at least once. Note that this cannot be verified without your typescript file.
- 2 points: username.txt contains name, username, and github username as specified
- 1 point: writeup.txt contains name, hobby, and song as specified.
- 1 points: Hello.java prints "Hello, " followed by the 0th command-line argument, followed by "!"

Acknowledgments

This lab is based on materials developed and refined by Tanzima Islam, Brian Hutchinson, Filip Jagodzinski, Qiang Hao, Nicholas Majeske, and several past TAs.