Computer Science 145

Lab 1 (10 points)

Due at 11:59 pm the night before your scheduled lab time.
Read all of the instructions. Late work will not be accepted.

The script Command

One of the commands you'll be using in this lab is the `script` command. The `script` command is used to record a terminal session, capturing all the input and output that occurs while the command is running. This is useful for keeping a record of your commands and their output, which can be helpful for debugging or documentation purposes.

When you run the `script` command, it starts a new shell session and begins recording everything that happens in that session. The recording continues until you exit the session, usually by typing `exit` or pressing `Ctrl+D`. By default, the recorded session is saved to a file named `typescript`, but you can specify a different filename if you prefer.

Details

Please complete the following steps. Before you begin running these commands in a terminal, run the `script` command to keep a log of your commands. You will need to turn in the transcript of your commands. The expectation is that you are completing this lab on the school lab machines. However, if the command is not found (e.g., you're in git bash), then you'll need to copy and paste your terminal contents at the end of the lab (Just do not clear your terminal as you work).

Basic git operations

1. One time only, before you use Git the first time, you need to tell it a little bit about yourself. 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
   ```

2. To improve security, Github now requires use of a Personal Access Token (PAT) in lieu of your password when accessing Github via the command line. Please create a PAT according to the following guide and securely store it for future reference: `https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/creating-a-personal-access-token` As for scopes/permissions, you should only need the repo box checked (and its three sub-boxes).

3. The first time you use Git for 145, you will also need to choose a location for your local repository. Let's assume we choose `~/145repo`. Conveniently, I have already created your main repository for 145 on Github. You can create a working copy and local repository linked to the main repository with the following command:

   ```
   # replace username with your wwu username
   git clone https://github.com/sizemore-teaching/202510_csci145_username ~/145repo
   ```

   Example: If your username is johndoe, your repository URL will look like this: ***`https://github.com/sizemore-teaching/202510_csci145_johndoe`***

4. Make a directory for `lab1` (assuming `~/145repo` is your local repository):

```
cd ~/145repo
mkdir lab1
```

5. Let's create an add a writeup file (spelling, spacing and capitalization matter):

```
cd lab1
touch writeup.txt # creates an empty file
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.

6. Commit your change to your 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 (e.g. `~/145repo` but not in the original Github repository. You can confirm this by browsing to the Github URL for your repository: `https://github.com/sizemore-teaching/202510_csci145_username`

7. Optionally, you may wish to configure the git command line tool to cache your Github credentials temporarily so you need not keep entering them. You can do so with a variant of the following:

```
git config credential.helper 'cache --timeout=3600'
```

   which will save your credentials for one hour (3600 seconds) – you can change the duration to your liking, trading off convenience for security. This configuration command need only be run once per repository, from within the working copy.

8. 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 check the Github URL again: you should see the directory and 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 statuses 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 adding` the writeup. Check the status again after committing, and then again after pushing.

13. Play around with the following commands (use `git help` or search the web for details):

    (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 writeup.txt` 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.

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.

Command-line Java

For this class, you'll be writing code in Java. As a first exercise, create the quintessential `HelloWorld` Java program, and add it to your `lab1` directory on your **main** branch.

1. Use an editor of your choice to create a `HelloWorld.java` program that prints `"Hello World"` to the screen.

2. To make sure everything works, compile and run your program from the command line:

```
javac HelloWorld.java
java HelloWorld
```

3. Add, commit, and push your `.java` file only, to your `main` branch in your `lab1` directory. By convention, compiled code, nor object files, are NEVER (except in special, rare cases, and this is not one of them!) pushed to a git repo.

```
git add HelloWorld.java
git commit -m "adding HelloWorld"
git push origin lab1
```

Using `.gitignore`

The `.gitignore` file is a special file used by Git to ignore or exclude certain files or directories from being tracked or added to the repository. This is useful for preventing temporary, private, or generated files (such as compiled code or editor backup files) from being included in your commits. (Here is a useful repo of `.gitignore` templates: `https://github.com/github/gitignore`).

1. Create a `.gitignore` file in the root of your repository (e.g., `~/145repo`):

```
cd ~/145repo
touch .gitignore
```

2. Open the `.gitignore` file with your preferred text editor and add the following lines to specify the types of files you want to ignore. For example, you can ignore all `.class` files like this:

```
# Ignore all .class files
*.class
```

3. Save and close the `.gitignore` file. From now on, any files matching the patterns specified in this file will be ignored by Git and will not be included in your commits.

4. Add and commit the `.gitignore` file to your repository:

```
git add .gitignore
git commit -m "Added .gitignore file"
git push
```

By using a `.gitignore` file, you can keep your repository clean and focused on the important files, while avoiding clutter from unnecessary or temporary files.

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's `lab1` directory if it isn't already there, and then `git add` it, commit and push. Confirm that your `lab1/typescript` file is visible by checking your github repo URL: `https://github.com/sizemore-teaching/202210_csci145_username`

Grading

To earn credit for this assignment you must

- correctly complete all of the steps, and
- have added, committed and pushed the `typescript` file documenting your work, and
- have your `writeup.txt` with the three lines described earlier in this document in your github repository, and
- have your `HelloWorld.java` file, compiles and runs as expected, and
- have a `.gitignore` file included at the root of your github repository.

The "Basic git operations" part is worth 5 points; "Branching and merging" is worth 3; "`HelloWorld.java`" is worth 1; and "`.gitignore`" is worth 1.

Acknowledgments

Thanks are owed to Brian Hutchinson, Filip Jagodzinski, Tanzima Islam, Qiang Hao, and several past TAs for contributions to the lab on which this lab is modeled.