SIENAcollege
Computer Science

Computer Science 330
Operating Systems
Siena College
Fall 2020

Lab 1: GitHub and Unix

Due: 11:00 AM, Friday, August 28, 2020

In this lab, you will learn or review, as the case may be, how to interact with a Unix system through the command line. In particular, you will work with Git and GitHub and compile a simple C program on a FreeBSD server.

You must work individually on this lab.

Learning goals:

1. Gain basic familiarity with the Unix command line.
2. Learn to interact with Git and GitHub from the Unix command line.
3. Learn how to edit, compile, and run C programs from the Unix command line.

Getting Set Up

You will receive an email with the link to follow to set up your GitHub repository, which will be named `unix-lab-yourgitname`, for this lab.

Trying out GitHub and noreaster

Before you can do this task, you need to have an account on noreaster and know your username and password. You also need to have the GitHub classroom link (sent by email) to be able to create your repository for this lab's work.

Log into your account on `noreaster.teresco.org` with Secure Shell. From Windows, one way to do this is to use PuTTY. From a Mac, a Windows Git Bash window, or a Unix/Linux system, you can use the `ssh` command in a terminal window. Once successfully connected, you should see a prompt that looks something like this:

```
[jcool@noreaster ~]$
```

Next, create a directory (that's our grown up computer scientist Unix word for what Microsoft and Apple want you to call a "folder") for your work for this course. If you want to name it "`opsys`", use the command:

```
mkdir opsys
```

Then change your working directory to that new directory:

```
cd opsys
```

Now, create your GitHub repository for this lab. To do so, open a browser window at `https://github.com` and sign in. In the same browser, visit the link you received in your email. This should bring you do a page where you can "Accept the **unix-lab** assignment". Do so. If that worked, the next page will have a link to a GitHub URL which is your copy of this lab's repository. Follow it.

You are now on the main page for your repository.

Let's add your name to the `README.md` file in the repository on GitHub. Click on its name, then in the window that comes up, click on the little pencil icon near the "Raw", "Blame", and "History" buttons to bring up GitHub's simple editor.

You'll be editing in a simple markup language called Markdown (please click here and read about it). Add your name to the file, then scroll down to the bottom and "Commit changes". You can leave the radio buttons above that set to "Commit directly to the master branch." Once you've commited your changes, you should now see the file displayed with your additions.

To get back to the main view of your repository, click on the "<> Code" tab near the upper left of the window.

Next, click on the green "Code" dropdown button and copy the URL that pops up. Go back to your terminal window on noreaster and type

```
git clone ...
```

replacing `...` with the URL you copied from the web page.

When prompted, log in with your GitHub credentials. You should then see some messages, and a new directory will be created. Change into that directory with the `cd` command, then list the contents with `ls`. You should see two files: `README.md` and `hello330.c`, which are the same two files that were in the repository on the GitHub site.

Before going further, we have a bit of configuration to do for your `git` setup on noreaster. Issue these commands (replacing with your own GitHub username and your email, of course) at the terminal prompt:

```
git config --global user.name "jcool"
git config --global user.email "jcool@teresco.org"
```

One of those files is a C program, so let's compile and run it on noreaster. You should be in the directory with the `README.md` and `hello330.c` files. First, let's take a look at the C program. You can see its contents with the command

```
cat hello330.c
```

We'll modify the file in a minute, but first let's compile and run this version. At your command prompt, type

```
gcc hello330.c
```

If this is successful, you should just get your prompt back in a second or so. This is typical of Unix: if a command is completed successfully, you will not get any output. "No News is Good News."

Run the `ls` command, and notice a new file listed: `a.out`. This is an executable file. To execute it, you can issue the command

```
./a.out
```

If all goes well, you should see the "`Hello, CSIS 330 world!`" printout.

Now, make a couple changes to the C program. Launch the Emacs editor to make those changes.

```
emacs hello330.c
```

For now, just use the arrow keys to navigate around (the mouse won't work, and the text-based menus at the top are more trouble than they're worth) and put your names in the comment at the top, and add at least one more `printf` statement to print additional messages. When you're done, type Ctrl-x then Ctrl-s to save your file, then Ctrl-x then Ctrl-c to exit Emacs. (3 points)

Recompile and re-run your program to make sure it works. If it does, run it one more time, but this time capturing its output in a file: (2 points)

```
./a.out > hello330.txt
```

If you list the files in your directory, you'll now see `hello330.txt` as well. You can see its contents with the `cat` command.

So you now have two copies of the repository, one on the GitHub server (the "origin" repository) and one in your account on noreaster (a "clone"). Later in the semester, you might have multiple clones in various places (noreaster, lab computers, maybe your own computer) and will want to be able to keep them in sync. Your clone has changes (2 new files, and one modified file) that are not in the origin on GitHub.

`git` knows what's changed, so let's ask it:

```
git status
```

You should see something like this:

```
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:   hello330.c

Untracked files:
  (use "git add <file>..." to include in what will be committed)

a.out
hello330.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

This means there is one file that `git` "knows about" that has changed: `hello330.c`, and there are two files that `git` doesn't know about: `a.out` and `hello330.txt`.

Since `a.out` is an executable file that gets generated from the C file, we don't want to store that in our repository. We'll ignore it for now. But the `hello330.txt` file is one we want to put in the repository (only because it's part of the grading). We can tell `git` to start tracking it with

```
git add hello330.txt
```

If you rerun the `git status` command, it should now be listed as a new file.

Once we're happy with a change or set of changes to files in our repository, we need to *commit* those changes to the repository. In this case, we will issue a command for each file, with an appropriate message in each:

```
git commit -m "Added name and extra printout." hello330.c
git commit -m "Required output capture." hello330.txt
```

Now a `git status` command should show something like this:

```
On branch master
Your branch is ahead of 'origin/master' by 2 commits.
  (use "git push" to publish your local commits)
Untracked files:
  (use "git add <file>..." to include in what will be committed)
```

```
a.out

nothing added to commit but untracked files present (use "git add" to track
```

Your changes have been committed to the clone, but they are not yet back up on the origin repository on GitHub. To do that, you will use

```
git push
```

This is short for "`git push origin master`", which says push the changes from this repository to the origin on the master branch. So far, we have not created any branches, so everything is on the master.

If your `push` command is successful, you should now see your changes reflected on GitHub (in the web page).

Now let's make a change right in the GitHub repository and then pull it down to our clone on noreaster. Edit the `README.md` file again to include a bulleted list of the `git` commands you used in this lab. Use Markdown to make it look pretty or at least easily readable. When you are done, include a meaningful commit message and use the green button to commit your changes. (4 points)

Now the repository on GitHub has changes that aren't in your clone. Back on noreaster, we want to "pull down" the changes:

```
git pull
```

Now, your `README.md` on noreaster should include the changes you made on GitHub.

More Unix Command Line

GUIs are nice, but they can be slow to navigate and too restrictive for some purposes. When logged into noreaster, in a Mac Terminal, or in a Git Bash window under Windows, you are working in a Unix environment and interacting with the system by typing commands at the Unix *shell*, or *command line*. When you log in or open one of these terminals, you will be presented with a prompt. This is your direct interface to issue commands to the operating system. When you type a command here, the shell will execute the command on your behalf, print out any results, then reissue the prompt.

Of course, the command line is useless if you don't know what commands it understands. You will learn about several important commands in this lab and many more throughout the semester. One of the most important is `man` – the Unix manual. Every Unix command has a manual page, including `man`. To see the manual page about `man`, type the command:

```
man man
```

Navigating the Directory Structure

Each program in a Unix system, including your shell, maintains the notion of a *working directory*. That is where the program will look for files unless instructed to do otherwise. You'll hear Unix users asking a question like "What directory are you in?" and the answer to this is your working directory.

When you first open a shell or connect to a server remotely, your *home directory* is your working directory. The command `pwd` will instruct the shell to print your working directory.

? Question 1:
What is your home directory on `noreaster.teresco.org`? (use `pwd` immediately after logging in) (1 point)

You can also list the contents of your working directory with the command `ls`.

? Question 2:
What output do you see when you issue the `ls` command on `noreaster.teresco.org`? (1 point)

Other important operations to navigate and modify the directory structure are changing your working directory (`cd`), creating a new directory (`mkdir`), and removing a directory (`rmdir`).

Create a directory in your account within your directory you created earlier for work in this course, but not inside the directory containing your clone of the git repository, to use for the for your work the next few minutes (`labstuff` might be a good name).

? Question 3:
Change your working directory to the one you just created and issue the `pwd` command. What does this show as your working directory? (1 point)

In your shell window and in your home directory (note: you can always reset your working directory to be your home directory by issuing the command `cd` with no parameters), issue this command:

```
uname -a > freebsd.txt
```

This will execute the command `uname -a`, which prints a variety of information about the system you are on, and "redirects" the output, which would normally be printed in your terminal window, to the file `freebsd.txt`.

Unix Commands

Identify the function of and experiment with these Unix commands (a few of which you have already used):

```
ls    cd      cp      mv      rm      mkdir   pwd
```

```
man    chmod    cat      more    grep    head     tail
ln     find     rmdir    wc      diff    tar      touch
```

Question 4:
Give a one sentence description of each command. (5 points total)

Using appropriate commands from the above list, move the `freebsd.txt` file you created in your home directory into the directory containing your clone of this lab's repository.

Show that this has worked by issuing the following command from inside of your course directory:

```
ls -laR > ls.out
```

Then move the file `ls.out` into your repository's directory. (1 point)

Use appropriate `git` commands to tell `git` to track these two new files (`freebsd.txt` and `ls.out`) and to get them into the repository's origin on GitHub. (2 points)

Submission

Commit and push!

Grading

This assignment will be graded out of 20 points.

Feature	Value	Score
Additions to `hello330.c`	3	
`hello330.txt`	2	
`git` command list	4	
Question 1	1	
Question 2	1	
Question 3	1	
Question 4	5	
`ls.out`	1	
Files on GitHub	2	
Total	20	