EECS 22: Assignment 1

Prepared by: Guantao Liu, Prof. Rainer Dömer

September 21, 2016

Due Thursday October 6, 2016 at 6:00 pm

1 Part1: Login to your Linux account

For this class, you will be doing your assignments by *logging in* to a shared machine (server) running the Linux operating system. Even though you may be using a personal computer or a workstation that is capable of computation locally, you will mainly be using them as *terminals* (clients), whose job is to pass keystrokes to the server and display outputs from the server.

To use a shared machine, first you need an *account* on the machine. EECS support has created an *account* for each student. To retrieve the username and password go to the following website:
`https://newport.eecs.uci.edu/account.py`.
The website asks for your UCInetID and the corresponding password before giving you the account information of your new EECS account. Note that your browser may also ask you to accept a certificate to open the secure website. If you have a problem please contact your EECS 22 TA (eecs22@eecs.uci.edu).

The names of the instructional servers are `zuma.eecs.uci.edu` and `crystalcove.eecs.uci.edu`. You can log into your account with your EECS username and password. Your account also comes with a certain amount of disk space. You can use this space to store homework assignment files, and you don't need to bring your own disks or other storage media.

1.1 Software and commands for remote login

You can connect to `zuma.eecs.uci.edu` or `crystalcove.eecs.uci.edu` from virtually any computer anywhere that has internet access. What you need is a client program for *remote login*.

Previously, people used **`rlogin`** or **`telnet`** to connect to the server, and **`ftp`** or **`rcp`** to transfer files. However, these protocols are insecure, because your keystrokes or output are in clear text and can be *snooped* by others. This means your account name and password can be stolen this way. So, for security reasons, do not use either of these programs.

Instead, use **`ssh`** as the primary way to connect to the server. **`ssh`** stands for *secure shell*, and it encrypts your network communication, so that your data cannot be understood by snoopers. For file transfers, use **`sftp`** or **`scp`**, which are secure.

Depending on what computer you use, it may have a different *implementation* of **`ssh`**, but the basic functions underneath are all the same. Check the course website on SSH:
`https://eee.uci.edu/16f/18030/resources.html`

- If you are logging in from a Windows machine, you can use **`SecureCRT`** or **`PuTTY`**.
- MacOS X already has this built-in (use Terminal or X11 to run a Linux shell). Most Linux distributions also bundle **`ssh`**.

- If you are logging in from an X terminal, you can use the command
  `% ssh zuma.eecs.uci.edu -X -l yourUserName`
  (note: `%` is the prompt, not part of your command) It will prompt you for your password. Note that the `-X` option allows you to run programs that open X windows on your screen.

1.2 Linux Shell

By now you should be logged in, and you should be looking at the prompt
`zuma% _`

Note: in the following writeup, we will show just
`%`
for the prompt, instead of
`zuma%`

You should change your password using the **`yppasswd`** command.

Try out the following commands at the shell prompt (See reference to the Linux Guide in section 1.3 for more details about these commands.).

`ls`	list files
`cd`	change working directory
`pwd`	print working directory
`mkdir`	make directory
`mv`	rename/move files
`cp`	copy files
`rm`	remove files
`rmdir`	remove directory
`cat`	print the content of a file
`more`	print the content of a file, one screen at a time
`echo`	print the arguments on the rest of the command line

Most commands take one or more file names as parameters. When referring to files, you may need to qualify the file name with directory references, absolute or relative paths:

`.`	current directory
`..`	one level higher
`~`	home directory
`/`	the root (top level) directory

1.3 Follow the Linux Guide

The best bet may be to search online for something like ”linux user tutorial,” ”linux user guide,” ”unix command line” or ”unix shell command” and check a few results to see what is agreeable to you. From those links, the following may be reasonable:
`http://linux.org.mt/article/terminal`
`http://www.linux-tutorial.info/modules.php?name=MContent&pageid=49`
or `ftp://metalab.unc.edu/pub/Linux/docs/linux-doc-project/users-guide/user-beta-1.pdf.zip`
(3.3.1-2, and chapter 4)
Learn basic shell commands: list files, change directory, rename files, move files, copy files, show file content.

There is nothing to turn in for this part.

1.4 Learn to use a text editor

There are three editors that are available on nearly all Linux systems that you may choose from.

pico is the easiest to get started with. A guide for **pico** can be found at:
`http://www.dur.ac.uk/resources/its/info/guides/17Pico.pdf`.

vi is a very powerful editor, but is arguably a bit more difficult to learn. Follow the **vi** guide at:
`http://www.ece.uci.edu/~chou/vi.html`

Finally, **emacs** is another editor that you may use. **emacs** is also a powerful editor, but is a bit easier to learn than **vi**. Follow the **emacs** guide at:
`http://www.gnu.org/software/emacs/tour/`.
Learn how to edit a file, move the cursor, insert text, insert text from file, delete words, delete lines, cut/paste, save changes, save to another file, quit without saving.

There is nothing to turn in for this part. However, it is critical that you get enough practice with your editor, so that you can do the homework for this class.

2 Part 2: Collatz conjecture (100 points)

The Collatz conjecture is a mathematical problem that still is not solved today. Please see a nice introduction of the problem and its missing mathematical proof on Wikipedia:
`https://en.wikipedia.org/wiki/Collatz_conjecture`

Rather than solving the problem by proper mathematical proof, we want to write a C program that calculates the Collatz sequence for any number input by the user.

Specifically, the program will ask the user to input a positive number which will serve as the initial value of the Collatz sequence (a_0). The program will then perform the Collatz function step by step until the terminating value 1 is reached.

For each step, the program should print the current value a_i on the screen. After the final value 1 is reached, the program should also print the number of steps (i) needed.

As an example, the interface of the C program should look like the following:

```
=======================
Collatz Conjecture Test:
=======================

Enter a positive integer (or 0 to quit): 5
Collatz sequence: 5, 16, 8, 4, 2, 1
Collatz conjecture is true for n=5.
Collatz sequence stops after 5 steps.

Enter a positive integer (or 0 to quit): 6
Collatz sequence: 6, 3, 10, 5, 16, 8, 4, 2, 1
Collatz conjecture is true for n=6.
Collatz sequence stops after 8 steps.

Enter a positive integer (or 0 to quit): 7
Collatz sequence: 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1
Collatz conjecture is true for n=7.
Collatz sequence stops after 16 steps.

Enter a positive integer (or 0 to quit): 0
Quit.
```

Since the Collatz conjecture is not proven, we need to ensure our program terminates even if the Collatz sequence would be infinite. To satisfy this condition for our algorithm, we will impose a maximum of 1 million steps. Then, if the sequence still has not terminated, the program should exit with the following message:

```
Stopping after 1 million steps.
Collatz conjecture not clear for n=XXX.
```

You should submit your program code as file **`collatz.c`**, a text file **`collatz.txt`** briefly explaining how you designed your program, and a typescript **`collatz.script`** which shows that you successfully compile and run your program.

In your type script, please run your program for the initial values 6, 19, and 27.

2.1 Writing your code

First create a subdirectory named `hw1` (for homework one). Change into the created directory `hw1`. Then, use your editor to create a C file named `collatz.c`. Do not use a word processor and transfer or paste the content. The C file should state your name and exercise number as a comment at the top of the file.

Note: Please write your C code using the proper indentation (either tabs or white spaces).

2.2 Compiling your code

To test your program, it must be compiled with the **`gcc`** command. This command will report any errors in your code. To call **`gcc`**, use the following template:

```
% gcc sourcefile -o targetfile
```

Then, simply execute the compiled file by typing the following:

```
% ./targetfile
```

Note: Please compile your C code using **-ansi -Wall** options as below to specify ANSI code with all warnings:

Below is an example of how you would compile and execute your program for the Collatz conjecture:

```
% gcc collatz.c -ansi -Wall -o collatz
% ./collatz
program executes
% _
```

3 Bonus (5 points)

Extend the Collatz conjecture program so that the maximum value reached in the sequence is printed, together with its corresponding step number.

For the example of starting value 27, the Collatz program should print the following result:

```
Collatz conjecture is true for n=27.
Collatz sequence stops after 111 steps.
Maximum value is 9232 at step 77.
```

To submit, use the same files as in Part 2, i.e. **`collatz.c`**, **`collatz.txt`**, and **`collatz.script`**.

4 Submission

To submit your work, you have to be logged into the server `zuma.eecs.uci.edu` or `crystalcove.eecs.uci.edu`.

Here is a checklist of the files you should have for submission. In the `hw1` directory, you should have the following files in your linux account:

- `collatz.c`
- `collatz.txt`
- `collatz.script`

We do require these *exact* file names. If you use different file names, we will not see your files for grading. Now, you should change the current directory to the directory containing the `hw1` directory. Then type the command:
`% turnin22`
which will guide you through the submission process.

You will be asked if you want to submit each file. Type yes or no. If you type "n" or "y" or just plain return, they will be ignored and be taken as a no. You can use the same command to update your submitted files until the submission deadline.

Below is an example of how you would submit your homework:

```
zuma% ls  # This step is just to make sure that you are in the correct directory that contains  hw1/
hw1/
zuma% turnin22
============================================================
EECS 22 Fall 2016:
Assignment "hw1" submission for eecs22
Due date:  Thu Oct 6 18:00:00 2016
** Looking for files:
** collatz.c
** collatz.txt
** collatz.script
```

```
============================================================
* Please confirm the following:                            *
* "I have read the Section on Academic Honesty in the       *
* UCI Catalogue of Classes (available online at             *
* http://www.editor.uci.edu/catalogue/appx/appx.2.htm#gen0) *
* and submit my original work accordingly."                 *
Please type YES to confirm.  YES
============================================================
File collatz.c exists, overwrite?  [yes, no] yes
File collatz.c has been overwritten
Submit collatz.txt [yes, no]?  yes
File collatz.txt has been submitted
Submit collatz.script [yes, no]?  yes
File collatz.script has been submitted
============================================================
Summary:
============================================================
Submitted on Tue Sep 20 17:00:21 2016
You just submitted file(s):
collatz.c
collatz.txt
collatz.script
% _
```

NOTE: You may need to use change mode command **`chmod`** to give read permission to the turnin script for submission. Therefore, if you see the script cannot submit some of your files, use the following command to change the permission to `755`. You may change back the mode to the initial mode `700` after finishing the submission.

```
zuma% chmod -R 755 hw1
```

4.1 Verify your submission

This step is optional, but recommended. If you want to confirm which files you have submitted, call the following command:
% **`/users/grad2/doemer/eecs22/bin/listfiles.py`**

This command lists your submitted files. Don't worry if you submitted too many files. We will only look at the files with defined names (here: `collatz.c`, `collatz.txt` and `collatz.script`) and ignore other files.

5 Typescript

A typescript is a text file that captures an interactive session within the Linux shell. Very often you are required to turn in a typescript to show that your program runs correctly. To create a typescript, use the **`script`** command. Here is an example:

- Type the command
  % **`script`**
  into the shell. It should say
  `Script started, file is typescript`
  % _
  This means it is recording every key stroke and every output character into a file named "typescript", until you hit **`^D`** or type **`exit`**.

- Type some shell commands. But don't start a text editor!
- Stop recording the typescript by typing **`exit`**.
  ```
  % exit
  Script done, file is typescript
  % _
  ```
- Now you should have a text file named `typescript`. Make sure it looks correct.
  ```
  % more typescript
  Script started on Mon 29 Sep 2014 04:58:45 PM PDT
  ...
  ...
  ```

You should immediately rename the typescript to another file name. Otherwise, if you run **`script`** again, it will overwrite the `typescript` file.

Note: If you backspace while in script, it will show the `^H` (control-H) character in your typescript. This is normal. If you use **`more`** to view the typescript, then it should look normal.