EECS 10: Assignment 1

Prepared by: Prof. Rainer Dömer

June 26, 2017

Due Monday July 3, 2017 at 11:00pm

1 Login to your Linux account

For this class, you will be doing your assignments by *logging on* 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. You should be able to login using your regular UCInetID credentials. If you have any problems, please contact your EECS 10 TA: eecs10@eecs.uci.edu

The name of the instructional servers are `zuma.eecs.uci.edu` and `crystalcove.eecs.uci.edu`. You can log into your account with your EECS user name 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. In the follwing sections, server `zuma.eecs.uci.edu` is used to illustrate the instructions, but you can use either `zuma.eecs.uci.edu` or `crystalcove.eecs.uci.edu` to do your assignment.

1.1 Software and commands for remote login

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

We will 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 network traffic is safe and secure. For file transfers, you may use **`sftp`** or **`scp`**, which are secure as well.

Depending on what computer you use, it may have a different *implementation* of **`ssh`**, but the basic function underneath is the same. Check the resources tab on the course web site for details:
`https://eee.uci.edu/17y/18010/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% _
```

You should change your password using the **`passwd`** 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 vs. 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`
Practice 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.

2 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/manual/html_node/emacs/`.

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.

3 Part1: `Initials.c` (text book exercise 2.25, page 67) [20 points]

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 `initials.c`. The C file should state your name and exercise number as a comment at the top of the file.

Since we only want to focus on text editing and initial programming practice, the task of your first programming assignment is very simple. Your program shall print your initials in large upper-case block letters down the page. Construct each block letter out of the letter it represents.

The following is an example for one of the inventors of the C language, Brian W. Kernighan:

```
    BBBBBBBBBBBBB
    B     B     B
    B     B     B
     B   B B   B
      BBB   BBB

           WWWWWW
     WWWWWW
    W
     WWWWWW
    W
     WWWWWW
           WWWWWW

    KKKKKKKKKKKKK
          K
        KK KK
     KK      KK
    K          KK
```

3.1 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:

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

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

```
zuma% ./targetfile
```

Below is an example of how you will compile and execute the Initials program and how your output should look like:

```
zuma% gcc initials.c -ansi -Wall -o initials
zuma% ./initials


    BBBBBBBBBBBBB
    B     B     B
    B     B     B
     B   B B   B
      BBB   BBB

           WWWWWW
     WWWWWW
    W
     WWWWWW
    W
     WWWWWW
           WWWWWW

    KKKKKKKKKKKKK
```

```
          K
       KK KK
   KK       KK
  K            KK
```

A brief text file, `initials.txt`, must be submitted as well that explains what the program does and why you chose your method of implementation. For this homework a single sentence should be sufficient.

You also need to show that it works with your own test cases by turning in a typescript named `initials.script`. For instructions on how to create a typescript, see Section 5 Typescript at the end of this document.

To submit your work, you have to be logged in to one of the following servers, `zuma` or `crystalcove`.

Here is a checklist of the files you should have:

In the `hw1` directory, you should have the following files in your Linux account:

- `initials.c`
- `initials.txt`
- `initials.script`

We do require these *exact* file names (i.e. do *not* replace initials with your actual initials). 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:

```
zuma% ~eecs10/bin/turnin.sh
```

which will guide you through the submission process.

You will be asked if you want to submit the script file. Type yes or no. If you type "n" or "no" 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:

4 Submit your work

```
zuma% ls # This step is just to make sure that you are in the correct directory that contains hw1/
hw1/
zuma% ~eecs10/bin/turnin.sh
=========================================
EECS10 Summer Session I 2017:
Assignment "hw1" submission for doemer
Due date:  Mon Jul 3 23:00:00 2017
** Looking for files:
** initials.c
** initials.txt
** initials.script
...
==============================================================
* Please confirm the following:                              *
* "Following the Academic Honesty Policy at UCI,             *
* I submit my own original work."                            *
   Please type YES to confirm.  YES
==============================================================
Submit initials.c [yes, no]?  y
File initials.c has been submitted
Submit initials.script [yes, no]?  y
File initials.script has been submitted
```

```
Submit initials.txt [yes, no]?  y
File initials.txt has been submitted
...
==========================================
Summary:
==========================================
Submitted on Mon Jun 26 10:20:52 2017
You just submitted file(s):
initials.c
initials.script
initials.txt
zuma% _
```

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:

```
zuma% ~eecs10/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: `initials.c`, `initials.txt` and `initials.script`) and ignore other files.

5 Typescript

A typescript is a text file that captures an interactive session with the Linux shell. To create a typescript, use the **`script`** command. Here is an example:

- Type the command
  ```
  zuma% script
  ```
  into the shell. It should say
  ```
  Script started, file is typescript
  zuma% _
  ```
  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`**.
  ```
  zuma% exit
  Script done, file is typescript
  zuma% _
  ```

- Now you should have a text file named `typescript`. Make sure it looks correct.
  ```
  zuma% more typescript
  Script done on Mon 23 Jun 2014 11:32:28 AM 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 hit 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.

6 Part 2: Adding Timestamps [30 Points]

Write a C program that computes the sum of two timestamps. Your program should prompt for the hours, minutes, and seconds for 2 timestamp values, then display the results. It should also display each timestamp after it has been entered.
When you run your program, it should look like this:

```
Please enter two timestamp values h1, m1, s1, h2, m2, and s2.
h1: 5
m1: 31
s1: 17
Timestamp 1 is 5 hours, 31 minutes, and 17 seconds.
h2: 3
m2: 40
s2: 59
Timestamp 2 is 3 hours, 40 minutes, and 59 seconds.
The sum is 9 hours, 12 minutes, and 16 seconds.
```

Note that the values for minutes and seconds must carry over if greater than 59.
Hint: Convert and store each timestamp triple as seconds before the addition in order to properly handle carryover in your computation. After you obtain the sum, convert it back to a timestamp triple so that it can be displayed in the desired format. You will need to use the * (multiplication), / (division), and % (modulo) operators for this.
The files that you should submit for this assignment are:

- **timestamp.c**: the source code file.
- **timestamp.txt**: the brief text file describing the design of your program, i.e. the steps required for the calculation.
- **timestamp.script**: the typescript file to show that your program works with the following 2 timestamps:
  5 hours, 31 minutes, and 17 seconds, and 3 hours, 40 minutes, and 59 seconds.

6.1 Bonus [5 Points]

Extend Part 2 above to also handle days and weeks (in addition to hours, minutes, and seconds).
You can use the same files as in Part 2. To demonstrate your program extension in the script file, add 2 weeks and 3 days, and 1 week and 6 days to the above timestamps, respectively.

6.2 Submission

Submission for these files will be the same as Part1.