EECS 10: Assignment 1

Prepared by: Weiwei Chen, Prof. Rainer Dömer

June 25, 2012

Due Monday July 2, 2012 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. 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 according 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 10 TA, (eecs10@eecs.uci.edu).

The name of the instructional server is `ladera.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.

1.1 Software and commands for remote login

You can connect to `ladera.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 function underneath are all the same. Check out OIT(NACS)'s page on SSH:
`http://www.nacs.uci.edu/support/sysadmin/ssh_info.html`
or check the course web site:
`https://eee.uci.edu/12y/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 ladera.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
`ladera% _`

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 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`
`ftp://metalab.unc.edu/pub/Linux/docs/linux-doc-project/users-guide/user-beta-1.pdf.zip` (3.3.1-2, and chapter 4)
`http://www.broadbandexpert.com/guides/ultimate-linux-guide/`
or
`http://www.nacs.uci.edu/help/manuals/uci.unix.guide/`
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.

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.nacs.uci.edu/help/manuals/uci.unix.guide/the_vi_editor.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.nacs.uci.edu/help/manuals/uci.unix.guide/editing_with_gnu_emacs.html`.

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: Exercise 2.25 (text book, 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`. 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. Write the C program according to exercise 2.25 in the text book.

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:

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

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

```
ladera% ./targetfile
```

Below is an example of how you would compile and execute the exercise 2.25:

```
ladera% gcc initials.c -o initials
ladera% ./initials
program executes
ladera% _
```

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 `ladera`.

Here is a checklist of the files you should have:

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

```
ladera%  /ecelib/bin/turnin10
```

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 "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:

4 Submit your work

```
ladera% ls # This step is just to make sure that you are in the correct directory that contains hw1/
hw1/
ladera% /ecelib/bin/turnin10
```

```
============================================================
EECS10 Summer Session I 2012:
Assignment "hw1" submission for eecs10
Due date:  Mon Jul 2 23:00:00 2012
** Looking for files:
** initials.c
** initials.txt
** initials.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.  y
============================================================
Submit initials.txt [yes, no]?  y
File initials.txt has been submitted
Submit initials.script [yes, no]?  y
File initials.script has been submitted
Submit initials.c [yes, no]?  y
File initials.c has been submitted
...
============================================================
Summary:
============================================================
Submitted on Mon Jun 24 18:15:50 2012
You just submitted file(s):
initials.txt
initials.script
initials.c
...
ladera% _
```

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:
`ladera%` **`/users/ugrad/2004/fall/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. 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
  `ladera%` **`script`**
  into the shell. It should say
  `Script started, file is typescript`

```
ladera% _
```
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**.
```
ladera% exit
Script done, file is typescript
ladera% _
```
- Now you should have a text file named `typescript`. Make sure it looks correct.
```
ladera% more typescript
Script started on Fri 22 Jun 2012 03:37:23 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.

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. In addition, 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: 14
m1: 22
s1: 39
Timestamp 1 is 14 hours, 22 minutes, and 39 seconds.
h2: 21
m2: 59
s2: 27
Timestamp 2 is 21 hours, 59 minutes, and 27 seconds.
The sum is 36 hours, 22 minutes, and 6 seconds.
```

Summed 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 your sum, convert it back to a timestamp triple so that it can be displayed. You will need to use the * (multiplication), / (division), and % (modulo) operators.
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:
7 hour, 46 minutes, and 55 seconds, and 2 hours, 27 minutes, and 42 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 5 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.