Assignment 1 – Introduction to Linux (5%)

Introduction

Assignment 1 is the first of 10 assignments in this course, and it contributes 5% towards your final course grade. The assignment total is 10 marks. You should begin Assignment 1 in Module 1; it is due at the end of Module 2. Check your Course Schedule for the precise due date. Directions for submitting Assignment 1 to your Open Learning Faculty Member for grading can be found in the Assignments Overview tab. An assignment marking criteria follows at the end of this document.

Instructions

You need to be knowledgeable of a typical operating system to understand the functionality of how it works. Windows is not a typical operating system as it has both operating system characteristics and well as user functionality mixed together. To have a better understanding of what an operating system is we are going to do some simple work on a Linux system this term. Assignment 1 is a simple introductory tutorial on the basics of Linux, including some basic programming in Java and C. We'll be using the Linux machine for programming at times through the course, so it is important to a basic understanding of the Unix basics.

> You must submit Java, and C files and screen shots that show how your program works. Refer to the marking criteria at the end of this document.

UNIX/Linux

Developed in the early 1970s at Bell Labs, UNIX was created as an "open" operating system that provided end-users the ability to extensively customise it, as needed. Because of its portability and low distribution cost, UNIX became very popular with universities, researchers, and businesses. It was eventually ported to most mainframe and mini-computers.

> *(Note: Although first coded in Assembly Language, UNIX was developed with the C programming language in mind. As long as a C compiler is available for a given platform (CPU and bus architecture), UNIX can be compiled for it!)*

For microcomputers, full UNIX is far too large (resource-wise). To run on these machines, different *flavours* of UNIX were introduced, such as FreeBSD, SCO-UNIX, XENIX, and Minix (Mini-UNIX). In 1991, Linus Torvalds took the Minix kernel and

crafted it for the i386 platform (based on the Intel 80386 CPU architecture, which includes all processors from 80386, 80486, and Pentium I/II/III/IV CPUs). He named the new, non-commercial flavour "Linux" (Little UNIX), and released it *freely* to the computing public for further development.

> *(Note: So why did Linus develop Linux? Most PC OSes were either not complex enough or too expensive (or both) for what he needed while going to school. The university's mini was shared by all students and faculty, with never enough available usertime, time for Linus to just "play." Wanting the power of the mini's OS at home...and the rest is history.)*

Today, many companies release their own Linux distributions ("distros") that include special configurations, software collections, and programming languages, with drivers for most new buses and devices (USB, IEEE1394, scanners, LCDs, etc.).

For all practical purposes, Linux is essentially the same as UNIX. Mostly all usercommands, application programs, and general concepts apply to Linux as they do UNIX. Aside from some administrative differences, users are equally comfortable in either operating system.

> *(Note: Linux is part of the GNU Software Development (GNU"GNU is Not Unix") and most Linux open-source software is part of the GPL (GPL-"GNU Public License"). "Opensource"—the source code is open to all.)*

So, which is the best Linux distribution? With so many available, and each almost specialised for specific tasks and environments (only all-purpose, or "generic," releases get media attention), the question should be: *which Linux distribution best fits the current needs?* For the moment, the answer is a personal one.

CLI/GUI

UNIX is a *command-line interface* (CLI), with interaction accomplished through a "shell" (of which there are many) that behaves like DOS's command line (or Window's Command Prompt). Many UNIX commands are similar to DOS (since most DOS commands actually originated from UNIX). Directories function almost identically in UNIX as they do DOS, except for advanced security and the concepts of "symbolic links" and "ownership."

Not to be outdone by MacOS or Windows, UNIX incorporates a graphical shell called a *window manager* to provide GUI features. Window managers use XWindows: a library of graphic functions providing graphical interfaces similar to Windows. Sitting atop X-Windows, the window manager provides a similar user- and application-interface to Windows and MacOS.

Future Focus of UNIX/Linux

Most computing professionals view UNIX as a "workhorse operating system," meaning that it is dependable, stable, and strong, but not very pretty, and this description has carried over to Linux. Most servers on the Internet, and in large businesses, run a flavour of UNIX rather than another OS; even Microsoft maintains a set of *NIX-based machines for some web-services (although they don't advertise this fact, of course).

But in providing a comfortable, intuitive environment for the average, end-user, Linux requires much improvement for installation/set-up and interaction tasks (relating equally to *software* and *hardware*). With periodic updates to the kernel, GUIs (*window managers*), and methods for installing new hardware, Linux has recently found itself being prematurely promoted as a "desktop operating system" rather than just a server-class operating system.

So the *question* is: *will Linux replace Microsoft Windows and Apple MacOS as the common desktop operating system?*

Many *pro*-Linux groups enjoy dreaming such, but the reality is that Windows and MacOS will continue—at least for the short term. Yet Linux has changed how Apple and Microsoft design their operating systems. From gaining blatant X-Windows "look-and-feel" aspects, to more reliable operating system cores designed around *kernel-level interactions* rather than *linear-polling* structures, Apple and Microsoft have shifted towards the UNIX direction.

Topics of this assignment

The breadth of topics related to learning UNIX/Linux is far beyond this one exercises. The goal of this assignment is to introduce the *NIX-style (as it is called) of operating systems, and help summarise the important and essential aspects.

1. The *command shell* through a terminal window
2. Basic command syntax and purpose; access to built-in help.
3. Directories: creation, navigation, and destruction; copying/moving files in directories.
4. "Who's logged on?" and file/directory ownership.
5. Editing files and language compilers: creating simple C++ and Java programs.

Observations and Tasks

Part 1: Getting Started

To access the TRU Linux server:

a. Machine name is: **cs2.tru.ca**

b. Set port to: **2200**

c. Connect using username and password obtained from your Open Learning Faculty Member

Note: You may need to install additional software to access the Unix server.

- [**Windows Users**]: To connect to the TRU Linux server, install PuTTY and WinSCP following the instructions provided below**.**
- [**Mac Users**]: You can use the built-in terminal software on your Mac to connect to the TRU Linux server. You may use any file transfer program of your choosing to move files between your Mac and the TRU Linux server.
- [**Linux Users**]: Connect to the remote host using: *ssh -p 2200 -l [username] cs2.tru.ca.* Use the *sftp* program to connect to the TRU Linux server on port 2200.

Additional software required for Windows users:

1. PuTTY is an X Window client. A free download of PuTTY is available at: http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html.
2. WinSCP is a secure file transfer program. A free down load of WinSCP is available at: http://www.winscp.com/

Part 2: Terminals and User Interaction and Basic commands

1. To open a *terminal window* (also called a "console" or "shell window"), click the taskbar icon that looks like a computer monitor.
2. Command line interaction with Linux is done through a *shell*, which performs the same function as *MS-DOS Command Prompt* in Windows: to provide a "command line interface" to the user.
3. Common shells: Bourne (bsh), Korn (ksh), C-shell (csh), tcsh (from C-shell), and zsh (from Korn shell), with the most popular Linux shell being bash (Bourne again shell).

   Along with command interaction, shells provide complete programming languages (programs written in shell languages are called shell scripts).

Although not fully featured as common programming languages (C++, COBOL, Java, Pascal), shells have almost as much capability as other scripting languages such as VBscript or Javascript.

4. There is a small set of often-used commands that must become familiar to all users. These are related to file copying/moving, directory navigation, and logging in/logging out.
5. Obtain a listing of files in your directory, type: **ls** To see all files (including hidden files), type: **ls –a**

   For a longer, detailed display, type: **ls –al**

   (Some Linux even provide **dir** command, similar to DOS; *try it*.)
6. To check if another specific computer is available, use the Internet **ping** command, type: **ping** 192.162.22.110 (Guru), and www.tru.ca (TRU Webserver)

> **Note: <*CTRL-C*>** *can exit almost any command line application.*
>
> *If this doesn't work, try quitting by just pressing the "**Q**" **key**.*

Part 3: Built-in Help

1. Most new Linux users (called *newbies* or *noobies*) find Linux very confusing because so many commands must be learned: *which command to use*, and *what does it do*. *(this is a similar complaint of* DOS*)*

   Such as the strange sounding commands: **mv**, **grep**, and **chown**.
2. With hundreds of available commands (written by different people, over the decades of UNIX/Linux's history), confusion can set in quickly. To help, designers always include "online manuals" called **man** pages that index [almost] every available command.

   For programmers, man *pages also include descriptions of key C-functions for use in relating to the OS, such as* **scanf()** *and* **printf()**.
3. To use a man page for help, use the syntax: **man** *command* Use this method to get help on the commands: **date**, **grep**, and **top**.
4. Another help method is to ask for condensed help directly from the command (built-in help). The syntax is: *command* **--help**
5. Most Linux distributions also include the **info** command. This is a large interactive, summary document of important Linux commands.

Info is used *alone* or *with a command*, use the syntax: **info** *command.*

For newbies getting familiar with one of the help methods is a must.

6. If you have not done so already, obtain help for the commands you typed in **Part 2** (using one of the methods: **man**, **--help**, or **info**).

   Did these commands have at least built-in **--help** available?

Part 4: Command Piping and Redirection

1. At the command line data can transferred from one command to another, or redirect input/output from/to files. These concepts are available in most operating systems because of the necessary programming required at the command line level.

   "piping" – pipe the output of one command as input to another;
   "redirection" – redirect output of a command to a file, or file content as input to a command

2. As an example of this technique, the following use more than one command per line, moving data from one to another.

3. To locate all the files in the running Linux operating system, and load the output into a buffer that you can scroll through (or page up/down), type:

   locate / | less

   Press 'Q' to quit the less *command.*

4. Output can also be sent to a file, rather than to the screen. To store a listing of all the files in the current directory (or 'folder' for GUI users), type: **ls -a > files.txt**

5. To examine the contents of the file, type: **less files.txt**

6. For a sorted listing of the files in the current directory, type either of the following:

   a) **ls –a | sort | less** or

   b) **ls –a | sort > files.txt** , followed by: **less files.txt**

7. The screen might be a little messy at this point, and it would be nice to clear the screen. To clear the CLI screen, type: **clear** Applying Piping and the cal command

8. An interesting command is **cal**, which produces a calendar view of a year, or just a month, for the range of years *1-9999*.

9. Get some help on the **cal** command and produce a display showing which day of the week your birthday falls on in 2003, 2004, and 2055.
10. Determine the cal command to display only your birthmonth this year. Pipe the output to a file called, **mybirthday.txt**

    Use **less** to examine the **mybirthday.txt** file.

Part 5: Resource Identification and Environment Variables

1. A main concern for servers is the availability of resources—the fewer resources available, the lower the overall system performance.
2. The following commands display the status of the essential resources available on the server, try them,

   free –o – memory allocation on the computer
   df – free disk space on main partitions (shown as devices)
   top – top running processes (measure of CPU utilisation); "q" to quit
   date – current date/time (not a resource, but useful)

3. Each login session obtains its own configuration environment. To display the current environment variables, in sorted order, type: **printenv | sort | less**
4. From this display, look up the values of the these environment variables,

   home - path for user's home directory
   hz - length of command history (previous commands used)
   language - CLI interface language (keyboard, display characters)
   logname - login name (default in this case is "knoppix")
   oldpwd - previous current directory (see **pwd**)
   path - directory path names to search for executing commands
   pwd - path of current working directory (where you are *now*)
   shell - directory path location for active shell program files
   user - name of user logged in (usually similar to **logname**)

Part 6: Directories

1. Directories in Linux behave similarly to directories/folders in Windows, except that Linux provides "symbolic links" (virtual directories pointing elsewhere) and "owned directories" (belonging to particular users).
2. In Linux, all devices (drives) and every user directory "hang" off the **root directory** (/). Unlike Windows, Linux does not have drive letters, only directories, even different drives (or partitions) are treated as directories.

[The following steps guide you through a little directory exercise.]

3. In your current user-directory, create a new directory called "mydir," type:

mkdir mydir.

4. Produce a listing with both **ls** and **ls –l** to see the directory you created.

   (Note: As you work through the following, produce a listing after each command to see how things change: **ls –l** *)*

5. Use the VI line editor to create the following file. Type: **vi myfile.file**

   ```
   Type Shift I for Insert
   Put the following in the file
   Linux is really neat.  Linux is different.
   Linux is not Windows.

   Hit Esc key to get out of insert mode
   :w to write and then
   :q to exit the editor
   ```

6. Copy the file into **mydir**, type: **cp myfile.file mydir/myfile.file**
7. Move the current file to another name (i.e., rename the file), type: **mv myfile.file other.file**
8. Change into the new directory, type**: cd mydir**
9. Move the file from the subdirectory to the parent directory, type: **mv myfile.file ..**
10. Change back to the parent directory, type: **cd ..**
11. With the subdirectory now empty, remove it, type: **rmdir mydir**
12. Remove the two files you created, type: **rm *.file**

Part 7: Writing a small program in C

(Ensure you are not in a root shell; root user should never develop code.)

1. Except for dedicated languages (such as for databases), there exists a Linux compiler, or interpreter, for almost all programming languages.
2. With the popularity of UNIX (and its various flavours) for the last 25 years, C/C++ has become *the* standard language for most all OSes.
3. Open a text editor to begin a new file, type: **vi smallprogram.c**

4. Code the C program below. When finished, save and quit the editor.

```
#include <stdio.h>
#define MAX 40     // maximum name length

//function prototype
void display (char n[], int times);

//main function (does not need prototype) int
main (void)
{     char name[MAX];   // user's name
int loop=0;       // number of loops

    printf ("What is your name? ");
gets(name);          // get user's name
printf ("How many times shall I print it? ");
scanf ("%d",&loop);  // read no. of times

    display (name, loop);
return (0); } // end of
main()
//function display (n)ame so many (t)imes void
display (char n[], int t)
{    int i=0;    for
(i=0; i<t; i++)
printf ("%s ",n);

   printf ("\n"); }//
end of display()
```

5. Quit the editor and return to the command line window.
6. To compile the program and generate the executable, type:
   gcc smallprogram.c –o smallprogram.out

   (ignore the warning about **gets())**
7. If there are syntax errors, edit the file again and fix them. To run the program, type: **./smallprogram.out**

Performing a Screen Capture

8. When running a program, its output and user interactions can be captured and sent to a file. This file can be brought into an editor, and then printed.
9. The **script** command opens a temporary shell and begins capturing (echoing) the complete input/output user interaction to a file. To stop the capture, close the file, and exit the shell, type **exit**

   For example (user input is in **bold**),

```
_> script out.txt
Script started, file is out.txt
_> ./smallprogram.out
What is your name? Bob
How many times shall I print it? 3
Bob Bob Bob
_> exit exit
Script done, file is out.txt
```

10. Be warned that **script** captures <u>all</u> console input/output. For systems that use an ANSI-graphic enabled user-shell, the script contains all console graphic manipulations.

Part 8: Writing a small program in Java

1. Linux provides more than just older programming languages. New languages also exist in Linux, such as: Java.
2. Use vi to code the following, and save it as smallprogram.java

```
import java.*;
import java.io.*;

public class smallprogram
{
public static void main(String[] args)
{
   String opinion = " likes Linux! ";    char
first=' ', last=' ';   // user's initials    int
loop = 20;      // number of times to repeat

   try
```

```
   {
      System.out.print("Your first initial? ");
first = (char)System.in.read();
        System.in.read();

      System.out.print("Your last initial? ");
last = (char)System.in.read();
        System.in.read();
   }
   catch (IOException e)
   {
   }
   System.out.println("Repeating "+loop+" times.");

   for (int i=0; i<loop; i++)
      System.out.print(first+"."+last+"."+opinion);

   System.out.println();

} // end of main()
} // end of smallprogram
```

---

3. Quit the editor and return to the command line window.
4. To compile the program, type: **javac smallprogram.java**
5. If there are errors, return to the editor, and fix them. To run the program, type: **java smallprogram**

Part 9: Finishing Up

1. Copy the Java and C programs, and the screen capture of the running programs off the Linux machine and submit them with a cover page as assignment 1.
2. Logout and close all windows.

Assignment Marking Criteria	Weighting
No syntax error: All requirements are fully implemented without syntax errors. Submitted screen shots will be reviewed with source code.	/5
Correct implementation: All requirements are correctly implemented and produce correct results Submitted screen shots will be reviewed with source code.	/5
Total	/10