COEN 177L: Operating Systems Lab

Lab assignment 1 -

Unix/Linux Commands and Basic Shell Programming

Objectives

1. To learn Unix/ Linux
2. To use command line programs
3. To develop sample shell programs

Guidelines

Good knowledge in Unix/Linux based C programming is required for all labs. You are highly encouraged to use command line tools in developing, compiling, and running your programs. You may use editors like vi or gedit and gcc compiler. This lab is designed to teach you basics of the Linux technical environment, including tools, commands, and shell scripts.

Please pay attention to your coding style and follow good programming practices, if your program is not worth documenting, it probably isn't worth running.[1] Please follow the GNU coding standards available on: https://www.gnu.org/prep/standards/html_node/Writing-C.html.

Unix/ Linux

Linux is a Unix-like operating system, following the design principles of Unix monolithic kernel in process control, cpu scheduling, memory management, file systems, networking, and access to the peripherals. Please read details on https://en.wikipedia.org/wiki/Linux.

Unix/ Linux has a culture of distinctive art and powerful design philosophy since its inception in 1969. Software technologies come and go but Unix/ Linux remained dominant and continued to evolve on a wide variety of machines ranging from supercomputers and PCs to handheld devices and embedded networking hardware. C language is ubiquitous and a central technology of Unix/ Linux. It is very hard to imagine developing applications at the core system level without C. The POSIX, Portable Operating System Standard, the Unix API (Application Programming Interface) is used to write truly portable software that can run across a heterogeneous mix of computers. TCP/IP (which you will learn in this class) and Unix represent the core technologies of the Internet. For more details, see [2].

If you are not using the ECC, you may need to install Linux on your laptop computer, either as a bare operating system or as a virtual machine. For details, please check https://www.linux.com. If you are using Mac OS, you can use Linux commands in a terminal window, including vi and gcc compiler.

Command-line programs (utilities)

The use of command-line programs is a tradition in Unix\Linux. Commands are executable programs that can run with variety of arguments (options). Command-line options are single letters preceded by a single hyphen, including:

-a: all, -b: buffer, -c: command, -d: debug, -e: execute, -f: file, -l: list, -o: output, -u: user

Some of the basic commands are:

- ls: lists all files and directories (try with options: -a, -al)
- cat: displays file content (try cat file1 file2 > file3)
- mv: moves a file to a new location (try mv file1 file2)
- rm: deletes a file
- cp: copy file

---

[1] Jonathan Nagler, "Coding Style and Good Computing Practices", *PS: Political Science and Politics*, Volume 28, Issue 3, 1995, pp. 488-492.

[2] Eric S. Raymond, *The Art of Unix Programming*, Pearson, 2004

- cmp: compares two files
- man: gives help information on a command
- history: gives a list of past commands
- clear: clear the terminal
- mkdir: creates a new directory
- cd: changes directory
- rmdir: deletes a directory
- chmod: changes the access mode of the specified files to the specified mode
- chown: changes the owner of the specified files to the specified userid
- echo: writes arguments to the standard output (try echo 'Hello World' > myfile)
- df: shows disk usage
- apt -get: install and update packages
- mail -s 'subject' -c 'cc-address' -b 'bcc-address' 'to-address' < filename: sends email with attachment
- chown/ chmod: change ownership/ permission of file or directory
- date: show the current date and time
- ps: displays active processes
- kill: kills process
- sh: bourne shell – command interpreter (good to learn about shell programming)
- grep: searches for pattern in files
- Ctrl+c: halts current command
- Ctrl+z: stops current command and resumes with foreground
- Ctrl+d (exit): logout of current session
- man: displays the manual page of the specified command
- echo: prints on the screen

For more commands, please refer to Linux command quick reference[3].

Shell programming (aka scripting)

A shell program (aka script) is a text file (typically has .sh extension, but not required) that contains standard Unix and shell commands. It allows you to execute a series of commands in a shell program simply by running the shell program rather than typing all commands. Shell programs are interpreted not compiled. They are used to automate system administration tasks. In a shell program, you can use

1. comments,
2. variables,
3. conditional commands,
4. repeated actions of commands, and
5. functions.

Bourne Shell (bsh or sh) is used for this lab. Other Shells are C-Shell – csh, Korn Shell, Born Again Shell – BASH, Thomas C-Shell – tcsh.

1. Comments: Use the # character to signify comments. Anything after a # character until the end of the line is considered a comment and is ignored by the shell.

[3] https://www.oreilly.com/openbook/debian/book/appe_01.html

2. Variables: Use letters, numbers, and the underscore to define variable names is a shell program. The assigned values to variables are stored internally as strings. To use a variable, precede the name with $. The shell provides the following pre-defined shell variables that may be used to pass parameters to a shell script.

$0	the name of the shell program
$1 to $9	the first through ninth parameters
$#	the number of parameters
$*	all parameters passed represented as a single word with individual parameters separated
$@	all the parameters passed with each parameter as a separate word
$?	hold the exit status of the previous command
$$	the process id of the current process
$PATH	the value of the PATH environment variable
$HOME	the full path name of your home directory
$USER	your user name
$PWD	the current directory path

3. Conditional commands: Use if keyword for a conditional statement in any of the following arrangements:

```
if expression
then
  command-list
fi

if expression
then
  command-list
elif expression
then
  command-list2
fi

if expression
then
  command-list1
else
  command-list2
fi
```

Expressions (Boolean):

- Relational operators:
  -eq, -ne, -gt, -ge, -lt, -le
- File operators:

-f *file*	True if *file* exists and is not a directory
-d *file*	True if *file* exists and *is* a directory
-s *file*	True if *file* exists and has a size > 0

- String operators:

-z *string*	True if the length of *string* is zero
-n *string*	True if the length of *string* is nonzero
s1 = *s2*	True if *s1* and *s2* are the same
s1 != *s2*	True if *s1* and *s2* are different
s1	True if *s1* is not the null string

- Compound comparison:

-a	And
-o	Or
!	Not

The other type of conditional command supported by the shell is the case statement (terminated by esac). The case command is similar to the switch-case construct in C, allowing the user to compare a single value against multiple values, and when a match is found, execute the corresponding command list.

4. Repeated actions of commands: The Bourne shell provides three repeated action commands: for, while, and until as follows:

```
for variable in word1 word2 word3 ... wordn
do
```

```
  command-list
done

while command
do
  command-list
done

until command
do
  command-list
done
```

5. Functions: shell functions are generally defined in a file in the format:

```
functionName () {
        command-list
}
```

Important Notes:

- Numerical computation in the shell is very awkward, but some *integer* mathematical expressions can be performed by using the expr command, e.g. i=`expr $i+1` (note the usage of backticks). You may instead enclose expressions in brackets, with similar restrictions e.g. i=$[i+1]
- To run a shell script called `script.sh`, you may either run the `sh script.sh` at the command line (will attempt to interpret any text file), or, alternately, you may add `#!/bin/sh` at the top of the file, execute the command `chmod u+x script.sh` to flag the file as executable, then run the script directly using the command `./script.sh`
- Some shell metacharacters are:

\	escape, for example \c takes character c
`...`	runs enclosed command and replace with output
'...'	takes without interpreting contents
"..."	takes after processing $, `...` and \

Sample shell program

Demonstrate each of the following steps to the TA to get a grade on this part of the lab assignment

Step 1. Write the following shell program using vi, emacs, or an editor of your choice. Note that spacing in shell scripts is critical, adding or removing spaces will likely cause the program to be interpreted erroneously!

```
#Sample shell programs for Lab assignment
#!/bin/sh
echo Executing $0
echo $(/bin/ls | wc -l) files
wc -l $(/bin/ls)
echo "HOME="$HOME
echo "USER="$USER
echo "PATH="$PATH
echo "PWD="$PWD
echo "\$\$="$$
user=`whoami`
numusers=`who | wc -l`
echo "Hi $user! There are $numusers users logged on."
if [ $user = "salagtash" ] #Replace with your own username!
then
        echo "Now you can proceed!"
else
        echo "Check who logged in!"
        exit 1
fi

response="Yes"
while [ $response != "No" ]
do
        echo "Enter height of rectangle: "
```

```
        read height
        echo "Enter width of rectangle: "
        read width
        area=`expr $height \* $width`
        echo "The area of the rectangle is $area"

        echo "Would you like to repeat for another rectangle [Yes/No]?"
        read response
done
```

Step 2. Run the shell program using either the sh command or using ./ as described in Important Notes.

Step 3. Rewrite the program in Step 1. so that you compute also the area of a circle, then demonstrate steps 1 – 3 to the TA. Note that this will require multiplication of non-integer values, which cannot be handled directly by the shell or by the expr command. Use a search engine to find how this can be done.

When your program runs without errors or warnings, demo to the TA.

Requirements to complete the lab

1. Show the TA correct execution of the shell programs.
2. ~~Submit your answers to questions, observations, and notes as .txt file and upload to Camino~~ N/A to lab 1
3. Submit the source code for all your programs as .sh file(s) and upload to Camino.

Be sure to retain copies of your .sh and .txt files. You will want these for study purposes and to resolve any grading questions (should they arise)

Please start each program/text file with a descriptive block that includes the following information:

```
# Name:
# Date:
# Title: Lab1 -
# Description:
```