CSEN 177L: Operating Systems Lab
Lab 1: Unix/Linux Commands and Basic Shell Programming

Objectives

1. To learn Unix/ Linux commands
2. To use command–line programs (utilities)
3. To develop sample shell programs

Guidelines

Good knowledge of 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 the vi editor to write your programs and the gcc for compilation. This lab is designed to introduce you to the basics of the Linux technical environment, including tools, commands, and shell scripts.

Please pay attention to your coding style and good programming practices; if your program is not worth documenting, it isn't worth running.[1] Please follow the GNU coding standards available at: 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 the Unix monolithic kernel in process control, CPU scheduling, memory management, file systems, networking, and access to the peripherals. Please read the details at https://en.wikipedia.org/wiki/Linux.

Unix/Linux has had 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 various 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, and the Unix API (Application Programming Interface) are used to write 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].

You may install Linux on your laptop, 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, including the vi editor and the 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
- cmp: compares two files
- man: gives help information on a command
- history: gives a list of past commands

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

- 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].

In general, you will probably use the commands - ls, more, mv, rm, mkdir, rmdir, cd, cp, chmod, who, ps, kill, ctrl+c, cmp, grep, cat, and man – more often. The man helps you to learn about a specific command and its use in Linux. For example, $man cat displays the meaning and usage of cat.

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 it 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.

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

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

$0	the name of the shell program
$1 thru$9	the first thru to ninth parameters
$#	the number of parameters
$*	all parameters passed are represented as a single word with 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 username
$PWD	the current directory path

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

```
if [ expression ]
then
  command-list
fi

if [ expression1 ]
then
  command-list1
elif  [ expression2 ]
then
  command-list2
fi

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

case keyword can also be used to execute one of several lists of statements depending on the value of a variable.

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 command. The case command allows the user to compare a single value against multiple values, and when a match is found, execute the associated 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 as:

```
name () {                          add () {
  commands;          e.g.            echo $[$1 + $2]
}                                  }
```

Important notes:

- The shell itself is not good for numerical computation, but use can still use some mathematical expressions by using the keyword expr, e.g. i=‘expr $i+1’ or you may close expressions in brackets, e.g. i=$[i+1]. For more complex calculations, experiment with the bc (basic calculator) command.
- Use the man (manual) command to find information about other commands, e.g. man bc
- Shell scripts are *extremely* sensitive to the placement of spaces, e.g. i = ‘expr $i+1` (spaces next to the “=” sign) will not function correctly. Neither will if[condition] (no spaces around the “[“ and “]” characters).
- Use #!/bin/sh as the first line of a shell program to define the path of the command interpreter, then use chmod u+x <file_name> to make it executable, then run with ./<file_name>
- Alternatively, run a text file as a shell script with sh <file_name>
- Shell metacharacters are:
  - '...' takes without interpreting contents
  - “...” takes after processing $, `...` and \
  - \ escape, for example \c takes character c
  - `...` runs an enclosed command and replaces it with output (`, or backtick, is the key above tab)

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. [20%] Write the following shell program using the vi editor

```
#Sample shell program for the 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" ] #use your username instead of salagtash
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. [30%] Run the shell program by typing sh <YourProgram.sh>. When it runs without errors or warnings, write down your observations in detail.

Step 3. [50%] Write a shell program to compute the area and the perimeter of a circle with the value of the circle's radius taken as input from the user. Check the value to make sure it is a strictly positive number. Calculation of the area and the perimeter must be performed as functions.

Demonstrate steps 1 – 3 to the TA. Remember: a successful demo is 25% of the grade

When your program runs without errors or warnings, make a copy of the source file

Requirements to complete the lab

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

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

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

```
# Name: <your name>
# Date: <date of lab>
# Title: Lab1 – <task>
# Description: This program computes <complete description here>.
```