Experiment 1
UNIX and C Programming[1]

Objectives

- ✓ To learn some UNIX commands
- ✓ To learn how to write and compile a C program for UNIX.

Prelab Activities

- ✓ Read the manual and try to do the experiment yourself before the lab.

General Information

UNIX Basic: UNIX is an operating system which was first developed in the 1960s, and has been under constant development ever since. By operating system, we mean the suite of programs which make the computer work. It is a stable, multi-user, multi-tasking system for servers, desktops and laptops. UNIX systems also have a graphical user interface (GUI) similar to Microsoft Windows which provides an easy to use environment. There are many different versions of UNIX, although they share common similarities. The most popular varieties of UNIX are Sun Solaris, GNU/Linux, and MacOS X. The UNIX operating system is made up of three parts; the kernel, the shell and the programs.

The kernel: The kernel is the core of the UNIX operating system. Basically, the kernel is a large program that is loaded into memory when the machine is turned on, and it controls the allocation of hardware resources from that point forward. The kernel knows what hardware resources are available (like the processor(s), the on-board memory, the disk drives, network interfaces, etc.), and it has the necessary programs to talk to all the devices connected to it. As an illustration of the way that the shell and the kernel work together, suppose a user types rm myfile (which has the effect of removing the file myfile). The shell searches the file store for the file containing the program rm, and then requests the kernel, through system calls, to execute the program rm on myfile. When the process rm myfile has finished running, the shell then returns the UNIX prompt to the user, indicating that it is waiting for further commands.

The shell: The shell acts as an interface between the user and the kernel. The shell is a command line interpreter (CLI). It interprets the commands the user types in and arranges for them to be carried out. The commands are themselves programs: when they terminate, the shell gives the user another prompt. Users can use different shells on the same machine. UNIX system offers verity of shells like 1) Bourne shell 2) c shell 3) Korn shell 4) Bash shell (very powerful & recommended for use, Linux default shell)

History - The shell keeps a list of the commands you have typed in. If you need to repeat a command, use the cursor keys to scroll up and down the list or type history for a list of previous commands.

---

[1] **Ref:** Tushar B. Kute, University of Pune, India.

The UNIX file system: All the stored information on a UNIX computer is kept in a file system. Any time we interact with the UNIX shell, the shell considers us to be located somewhere within a file system. The place in the file system tree where we are located is called the current working directory.

The UNIX file system is hierarchical (resembling a tree structure). The tree is anchored at a place called the root, designated by a slash "/". Every item in the UNIX file system tree is either a file, or a directory. A directory is like a file folder. A directory can contain files, and other directories. A directory contained within another is called the child of the other. A directory in the file system tree may have many children, but it can only have one parent. A file can hold information, but cannot contain other files, or directories.

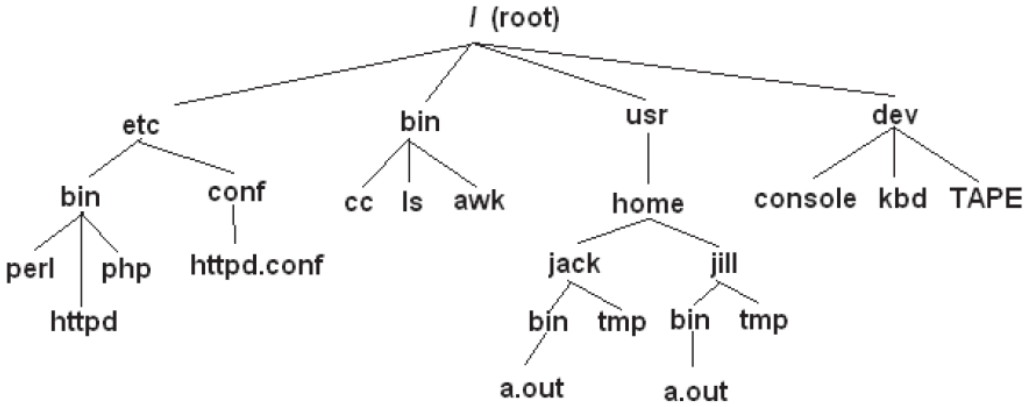

Figure 1. UNIX / Linux directory structure.

To describe a specific location in the file system hierarchy, you must specify a "path". The path to a location can be defined as an absolute path from the root anchor point, or as a relative path, starting from the current location. When specifying a path, you simply trace a route through the file system tree, listing the sequence of directories you pass through as you go from one point to another. Each directory listed in the sequence is separated by a slash. UNIX provides the shorthand notation of "." to refer to the current location, and ".." to refer to the parent directory.

Linux: Linux is not UNIX, as UNIX is a copyrighted piece of software that demands license fees when any part of its source code is used. Linux was written from scratch to avoid license fees entirely, although the operation of the Linux operating system is based entirely on UNIX. It shares UNIX's command set and look-and-feel, so if you know either UNIX or Linux, you know the other, too. Linux is a freely distributable version of UNIX developed primarily by Linus Torvalds at the University of Helsinki in Finland.

Text editors in Linux: Linux is just as well suited for word processing as any other operating system. There are several excellent word processing programs for Linux like AbiWord, KWord, part of the KOffice suite and the OpenOffice.org as well as StarOfficesuite's word processor.

Why use a text editor?: A text editor is just like a word processor without a lot of features. All operating systems come with a basic text editor. Linux comes with several. The main use of a text editor is for writing something in plain text with no formatting so that another program can read it. Based on the information it gets from that file, the program will run one way or another.

The text editor "vi": The most popular text editor for Linux is called 'vi'. This is a program that comes from UNIX. There is a more recent version called 'vim' which means 'vi improved'. The problem with 'vi' or 'vim' is that a lot of people don't like it. You have to remember a lot of key combinations to do stuff that other text editors will do for you more easily.

Let's make a text file. Type:

```
vi os.txt
```

We'll see a line of tildes down the left side and the name 'os.txt' at the bottom and [new file].

To write something, we have to press ESC and the '`i`' key (i for insert). We can always erase our mistakes with the backspace key.

To save this file, we would press ESC then the colon key '**:**' then '`w`' (write)

To save the file and quit vi, you would press ESC, ESC the colon key '**:**' then `wq` (write, quit)

To quit without saving, press ESC, '**:**' then '`q`'. vi may protest if we've written something and we don't want to save it. If we press ESC '**:**' '`q!`' with an exclamation point, vi will accept it and not save our changes.

'pico' and 'joe' are actually two other text editors like 'vi'.

Exercise 1 — UNIX Commands

You should open a terminal window and practice with the following commands.

UNIX Commands:
You interact with the UNIX operating system by entering a command at the shell prompt.

Help for a command usage To check the detail of a command (for example ls) type

```
man ls
```

where man is the command which displays the manual pages of the given command

Command syntax The basic form of any Unix command is:

```
command_name options argument(s)
```

Command options Options modify the way that a command works. They usually consist of a hyphen followed by a single letter.

For example the `wc` command counts the number of words, characters and lines in a file. By using a different option you can choose what is counted.

`wc –w file1`	counts the words
`wc -c file1`	counts the characters
`wc -l file1`	counts the lines

Using multiple options Some commands accept multiple options. These can be grouped together after a single hyphen:

```
command –abc argument
```

Location of commands Unix commands are executable binary files located in directories with the name **bin** (for binary). Many of the commands that you use are located in the directory **/usr/bin**.

When you enter the name of a command the shell checks to see if it is a built-in command. If it is not, it looks for the binary file that the command name refers to in each of the directories that are defined in the **PATH** environment variable.

Locating a command To find out if a command or utility is available on your system enter the command:

```
which command_name
```

If the command is found its pathname is displayed. The message *`command_name: Command not found`* *is* displayed when the command cannot be found.

Entering more than one command To enter several commands on the one command line, use a **;** (semicolon) to separate each one from the next. For example:

```
cd .. ; ls -l
```

This command line contains two commands. The first, **cd ..** changes the current directory to the parent directory. The second, **ls -l** produces a long listing of the contents of the current directory. If necessary you can continue the commands onto another line.

Using the command history Each time that you enter a command it is added to a command history list. You can reuse commands from this list. This command history facility is not available with the Bourne shell (sh).

To display the command history list enter the command:

```
history
```

This displays a numbered list of commands in the order in which you have used them.

Re-running a command from the history list To run a command from the history list, use a command from this table.

Run the previous command	**!!**
Run command number n	**!n**
Run most recent command starting with characters in string	**!string**
Run most recent command containing characters that match string	**!?string**

Changing the size of the command history By default only the previous command is saved in the command history list. To save a larger number of commands set the history variable in your shell start-up file. For example:

```
set   history=50
```

This will keep a history of the fifty (50) most recent commands that you have used.

Redirecting standard input and output

UNIX considers any device attached to the system to be a file. And that includes your terminal! By default, a command treats your terminal as the standard input file from which to read in information. Your terminal is also treated as the standard output file to which information is sent from the command. This action can be changed by redirecting standard input and standard output from and to any other file.

Redirecting standard input To redirect the standard input for a command use the **<** (less than) character followed by the name of the input file. For example:

```
mail tony < memo
```

This redirects the standard input to the **mail** command so that it comes from the file **memo**. The effect of this is to mail the contents of this file to the user tony.

Redirecting standard output To redirect the standard output from a command use the **>** (greater than) symbol followed by the name of the output file. If the file that you redirect standard output does not already exist it will be created. For example:

```
grep Smith /etc/passwd > popular
```

This redirects the standard output from the **grep** command so that it goes to the file **popular**. This file will contain all occurrences of the string Smith that were found in the `/etc/passwd` file.

Appending standard output to a file To append the standard output from a command to a file use two **>>** (greater than) symbols followed by the name of the file. If the file does not exist it is created. For example:

```
cat part1 > chapt2
cat part2 >> chapt2
```

This creates a file called chapt2 with the same contents as part1. It then reads the contents of part2 and appends them to the file chapt2. The file chapt2 now contains the data from part1 followed by the data from part2.

Connecting commands together UNIX allows you to link two or more commands together using a pipe. The pipe takes the standard output from one command and uses it as the standard input to another command.

```
command1 | command2 | command3
```

The **|** (vertical bar) character is used to represent the pipeline connecting the commands. With practice you can use pipes to create complex commands by combining several simpler commands together.

To pipe the output from one command into another command:

```
who | wc -l
34
```

This command tells you how many users are currently logged in on the system.

The standard output from the **who** command - a list of all the users currently logged in on the system - is piped into the **wc** command as its standard input. Used with the -l option this

command counts the numbers of lines in the standard input and displays the result on the standard output.

Some Other UNIX Commands:

`cd`

'`cd`' means 'change directory'.

Typing: `cd /[directory name]` will get us into one of the main directories.

Typing `cd ..` will get us out of it. (move to parent directory.)

Typing `cd` without the `/` and a sub-directory name will get into that subdirectory.

If we type just: `cd` we'll go back to our home directory.

`ls`

Typing '`ls`' will list the contents of a directory with just information about file names.

The syntax for the ls command is:

```
ls [options] [names]
```

Options:

`-a`	Displays all files.
`-b`	Displays nonprinting characters in octal.
`-c`	Displays files by file timestamp.
`-C`	Displays files in a columnar format (default)
`-d`	Displays only directories.
`-l`	Displays the long format listing.
`-L`	Displays the file or directory referenced by a symbolic link.
`-m`	Displays the names as a comma-separated list.
`-n`	Displays the long format listing, with GID and UID numbers.
`-o`	Displays the long format listing, but excludes group name.
`-p`	Displays directories with /
`-q`	Displays all nonprinting characters as ?
`-r`	Displays files in reverse order.
`-R`	Displays subdirectories as well.
`-t`	Displays newest files first. (based on timestamp)
`-u`	Displays files by the file access time.
`-x`	Displays files as rows across the screen.
`-1`	Displays each entry on a line.

Example:

```
ls -la
```

`cp`

'`cp`' is used for copying files from one place to another, or for making a duplicate of one file under a different name.

Example :

```
cp first.txt second.txt
```

The first.txt file will be copied into second.txt

mv

'mv' is usedfor moving files from one place to another. It cuts the file from one place and pastes it to another.

Options:

-f Forces the move.

-i Prompt for a confirmation before overwriting any files.

syntax :

```
mv [options] sources target
```

Examples:

```
mv -f sitrc /usr
```

It will move the file `sitrc.txt` to the directory `usr`

mkdir

This command is used for making or creating directories. The syntax for the `mkdir` command is:

```
mkdir [options] directories
```

options:

-m Sets the access mode for the new directory.

-p If the parent directories don't exist, this command creates them.

Examples:

```
mkdir SITRC
```

rmdir

This is the opposite of 'mkdir'- which is used to delete the directories. It should be pointed out that in order to use it, the directory has to be empty.

So, we have to use your 'rm' command for removing files in the directory.

rm

This command is used for removing or deleting files.

Syntax:

```
rm [options] filename
```

Options:

-r removes directories, removing the contents recursively beforehand (so as not to leave files without a directory to reside in) ("recursive")

-i asks for every deletion to be confirmed ("interactive")

-f ignores non-existent files and overrides any confirmation prompts ("force"), although it will not remove files from a directory if the directory is write protected.

more and less

'more' is a command that we can use to read, for example, what's written in a file. We would type `'more xyz'` to see the file completely. Then, we can press the 'q' key to stop viewing the file. We can scroll back up to see the whole text if we want.

grep

The `grep` command allows us to search one file or multiple files for lines that contain a pattern. Exit status is `0` if matches were found, `1` if no matches were found, and `2` if errors occurred.

The syntax for the grep command is:

```
grep [options] pattern [files]
```

options:

`-b` Display the block number at the beginning of each line.

`-c` Display the number of matched lines.

`-h` Display the matched lines, but do not display the filenames.

`-i` Ignore case sensitivity.

`-l` Display the filenames, but do not display the matched lines.

`-n` Display the matched lines and their line numbers.

`-s` Silent mode.

`-v` Display all lines that do NOT match.

`-w` Match whole word.

Examples:

```
grep -c tech file1
```

who

This is used to find out who's working on our system. As we now know, Linux is a multiuser system. Even if we're using one computer at our home, we may be working as more than one person. For example, if we logged in as '`root`' but are working as '`nitin`'. We may see something like this:

```
root tty1 May 20 09:48
nitin tty2 May 20 10:05
```

This is just Linux's way of saying that '`root`' started working on terminal 1 on May 20 at 9:48 in the morning and nitin started working on terminal 2 at 10:05. This is mainly used in networked situations so the system administrator knows who's working.

whoami

It is a little program that tells us who we are, just in case we didn't know already. Actually it tells us who we are in terms of how Linux understands who you are, that is to say, our user name.

Syntax:

```
whoami
```

pwd

(print working directory)

The pwd command displays the full pathname of the current directory.The syntax for the pwd command is:

```
pwd
```

cat

The cat command reads one or more files and prints them to standard output. The operator > can be used to combine multiple files into one. The operator >> can be used to append to an existing file.

The syntax for the cat command is:

```
cat [options] [files]
```

options:

-e	$ is printed at the end of each line. This option must be used with -v.
-s	Suppress messages pertaining to files that do not exist.
-t	Each tab will display as ^I and each form feed will display as ^L. This option must be used with -v.
-u	Output is printed as unbuffered.
-v	Display control characters and nonprinting characters

Examples:

```
cat file1
cat file1 file2 > all
cat file1 >> file2
```

wc

This command will give us the number of lines, words and letters (characters) in a file and in that order.

Usage:

```
wc -l <filename> print the line count
wc -c <filename> print the byte count
wc -m <filename> print the character count
wc -L <filename> print the length of longest line
wc -w <filename> print the word count
```

ps

It gives us a list of the processes running on our system. Just typing `ps` will give us the processes we're running as a user.

chmod

The chmod command changes the access mode of one file or multiple files. The syntax for the chmod command is:

```
chmod [option] mode files
```

options:

-R	Descend directory arguments recursively while setting modes.
-f	Suppress error messages if command fails.

mode:

who	u=user, g=group, o=other, a=all (default)

Opcode

+ means add permission

− means remove permission

= means assign permission and remove the permission of unspecified fields

Permission

```
r=Read, w=write, x=Execute
```

Examples:

```
chmod ug+rw mydir
chmod a-w myfile
```

Exercise 2 — UNIX C Programming

1. Write a simple C program Open any text editor, and write and save the following code.

The following program reads an integer number from the user and it prints all the positive odd integers which are less than the given integer.

odd.c

```
#include <stdio.h>
int main (void)
{
    int x, n;
    printf("Enter an Integer: ");
    scanf("%d",&n);
    printf("The list of all positive odd integers less than %d:\n",n);
    for(x=1;x<n;x++)
    {
        if (x%2 != 0)
            printf("%d\n",x);
    }
    return(0);
}
```

2. Compiling a C program Compile the source codes.
To compile a C program the following command is used

General format

gcc filename

Example

gcc odd.c

This will produce an executable file called a.out

If you want to get the executable file in your own name then the command for compiling will change as

gcc odd.c -o odd

Now this command will give an executable file in the name **odd**

3. Executing a C program Run the program.
To execute a C program the following command is used

General format

executablefilename

Example

odd

This will not work for some times, because of path settings. So, specify the current directory as path (relative path)

./odd