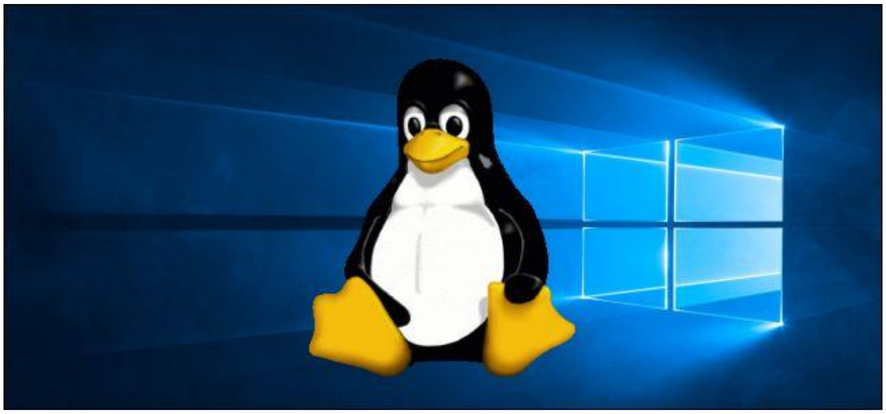
UNIVERSITY OF ENGINEERING AND TECHNOLOGY, TAXILA

FACULTY OF TELECOMMUNICATION AND INFORMATION ENGINEERING

SOFTWARE ENGINEERING DEPARTMENT



Operating Systems

Experiment 3

Implementation of LINUX Commands -II

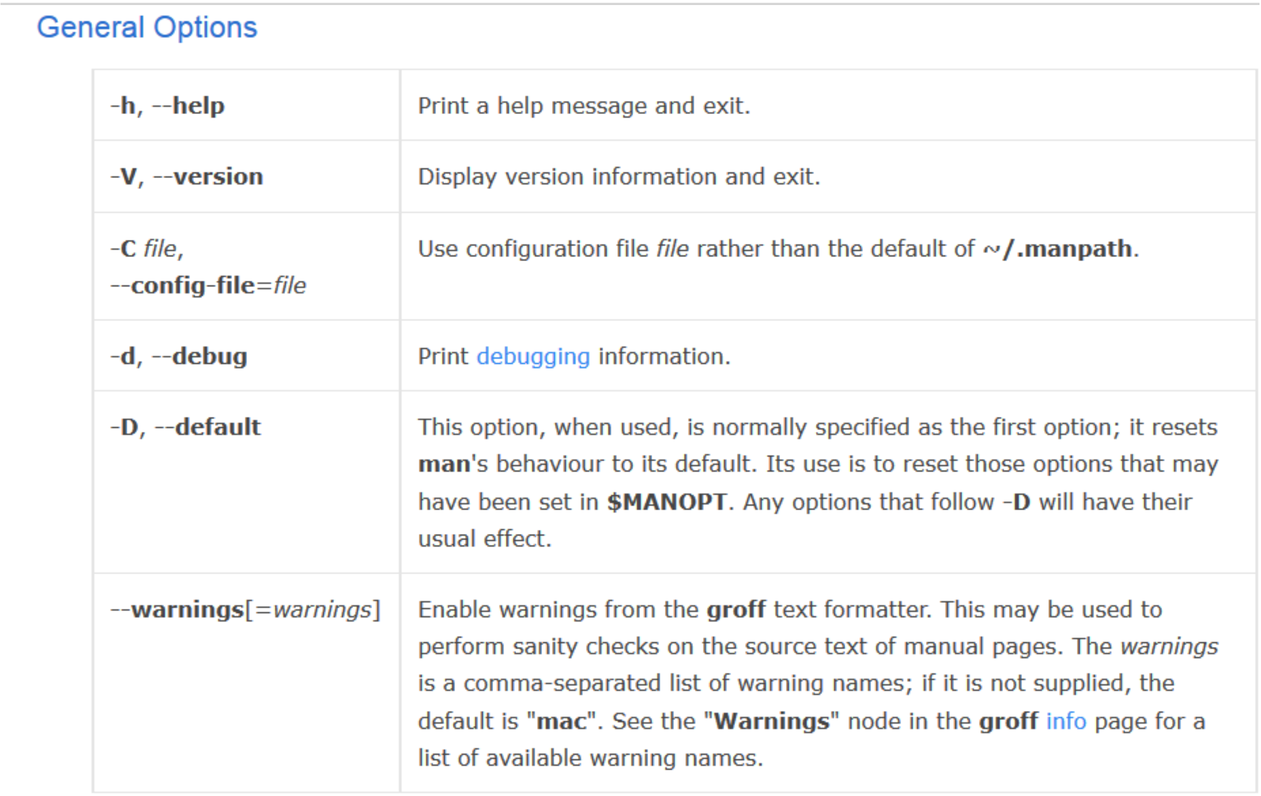
(Linux)

CLO 2. Use modern tools and languages.
CLO 3. Demonstrate an original solution of problem under discussion.
CLO 4. Work individually as well as in teams

Implementing Linux Commands

- ## HELP

$ man [command name]



General Options

-**h**, --**help**	Print a help message and exit.
-**V**, --**version**	Display version information and exit.
-**C** *file*, --**config-file**=*file*	Use configuration file *file* rather than the default of **~/.manpath**.
-**d**, --**debug**	Print debugging information.
-**D**, --**default**	This option, when used, is normally specified as the first option; it resets **man**'s behaviour to its default. Its use is to reset those options that may have been set in **$MANOPT**. Any options that follow -**D** will have their usual effect.
--**warnings**[=*warnings*]	Enable warnings from the **groff** text formatter. This may be used to perform sanity checks on the source text of manual pages. The *warnings* is a comma-separated list of warning names; if it is not supplied, the default is "**mac**". See the "**Warnings**" node in the **groff** info page for a list of available warning names.

- ## Date Time

$ date

Description: Prints the system date and time.

- ## Calendar

$ cal

Description: Prints an ASCII calendar of the current month

- **head**

Syntax: $ head [option] [Filename]

Description:

“head” displays the top part of a file.

By default it shows the first 10 lines.

-n allows you to change the number of lines to be shown.

Examples:

head –n 50 file.txt

Displays the first 50 lines of the file.txt

$head -18 filename

Displays the first 18 lines of the file called filename.

- **tail**

Syntax: $ tail [option] [Filename]

Description:
Display last 10 (by default) lines of a file.
Same as head command.
Example:
$tail -12 filename
Displays the last 12 lines from the ending

- **File Permissions**

Each file in UNIX/LINUX has an associated permission level. This allows the user to prevent others from reading/writing/executing their files or directories.

Syntax: ls –l [filename]
Description: To find permission level of the file.

The permission levels are:

- **“r”** means “read only” permission.

- **“w”** means “write” permission.
- **“x”** means “execute” permission.
- **In case of directory, “x” grants permission to list directory contents.**

- ## Command: chmod

If you own a file, you can change its permissions with “chmod”.

$ **chmod [user/group/others/all] +[permission] filename**

Let's say you are the owner of a file named **myfile**, and you want to set its permissions so that:

1. the **u**ser can **r**ead, **w**rite, ande **x**ecute it;
2. members of your **g**roup can **r**ead ande **x**ecute it; and
3. **o**thers may only **r**ead it.

This command will do the trick:

```
chmod u=rwx,g=rx,o=r myfile
```

This example uses **symbolic permissions notation**. The letters **u**, **g**, and **o** stand for "**user**", "**group**", and "**other**". The equals sign ("=") means "set the permissions exactly like this," and the letters "**r**", "**w**", and "**x**" stand for "read", "write", and "execute", respectively. The commas separate the different classes of permissions, and there are no spaces in between them.

Here is the equivalent command using **octal permissions notation**:

```
chmod 754 myfile
```

Here the digits **7**, **5**, and **4** each individually represent the permissions for the user, group, and others, in that order. Each digit is a combination of the numbers **4**, **2**, **1**, and **0**:

- **4** stands for "read",
- **2** stands for "write",
- **1** stands for "execute", and
- **0** stands for "no permission."

So **7** is the combination of permissions **4**+**2**+**1** (read, write, and execute), **5** is **4**+**0**+**1** (read, no write, and execute), and **4** is **4**+**0**+**0** (read, no write, and no execute).

Example 1:

```
chmod  7     7      7     filename
       user  group  others
```

Gives user, group and others r, w, x permissions

$chmod 750 filename

Gives the user read, write and execute.
Gives group members read and execute.
Gives others no permissions.
Using numeric representations for permissions:
r = 4; w = 2; x = 1; total = 7

- ## Redirection of Input and Output

Mostly all command gives output on screen or take input from keyboard, but in Linux (and in other OS’s also) it's possible to **send output to file or to read input from file.**
For e.g.
$ ls command gives output to screen; to send output to file of ls command give command

$ ls > filename
It means to put output of ls command to filename.

There are three main redirection symbols **>,>>,<.**

> Redirector Symbol (Overwrite)

Syntax: Linux-command > filename
Description: To output Linux-commands result (output of command or shell script) to file. Note that if file already exist, it will be overwritten else new file is created.

>> Redirector Symbol (Append)

Syntax: Linux-command >> filename
Description: To output Linux-commands result (output of command or shell script) to END of file. Note that if file exists, it will be opened and new information/data will be written to END of file, without losing previous information/data, and if file does not exist, then new file is created.

< Redirector Symbol

Syntax: Linux-command < filename

Description: To take input to Linux-command from file instead of key-board.

- ## sort

Example:

Create text file sname as follows

```
$cat > sname
virk
ash
zebra
babu
```

Press CTRL + D to save.

Now issue following command.

```
$ sort < sname > sorted_names
$ cat sorted_names
ash
babu
virk
zebra
```

In above example sort (**$ sort < sname > sorted_names**) command takes input from **sname fil**e and output of sort command (i.e. sorted names) is redirected to **sorted_names file**.

Try one more example to clear your idea:

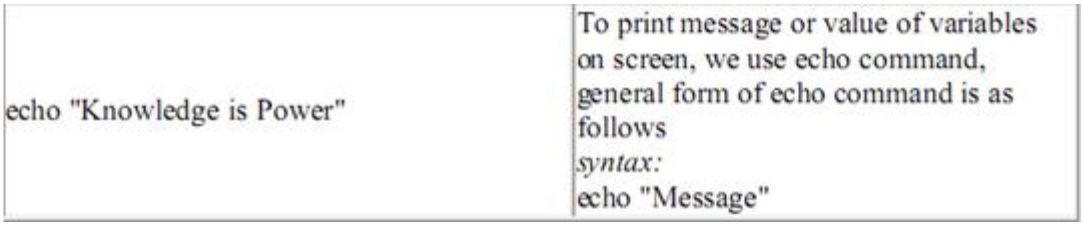
```
$ tr "[a-z]" "[A-Z]" < sname > cap_names
$ cat cap_names
VIVEK
ASHISH
ZEBRA
BABU
```

tr command

is used to translate all lower-case characters to upper-case letters. It takes input from sname file, and tr's output is redirected to cap_names file.

Shell Scripts

1. Open editor
2. Type code
3. save with .sh extension
4. run as sh lab3.sh



echo "Knowledge is Power"	To print message or value of variables on screen, we use echo command, general form of echo command is as follows *syntax:* echo "Message"

Type:

```
$ echo “The cost of the item is $15”
The cost of the item is 5
```

To display an actual dollar sign, you **must precede** it with a **backslash character:**

```
$ echo “The cost of the item is \$15”
The cost of the item is $15
```

Variables in Shell

A variable in a shell script is a means of **referencing** a **numeric** or **character value**. And unlike formal programming languages, a shell script doesn’t require you to **declare a type** for your variables. in Linux shell scripting we are using two types of variables: **System Defined Variables** & **User Defined Variables**.

1. System variables

These are the variables which are created and maintained by **Operating System (Linux) itself.** Generally, these variables are defined in **CAPITAL LETTERS**. We can see these variables by using the command “**$ set** “. some of the important *System variables are:* ***HOME, USER***
You can print any of the above variables contains as follows:

```
$ echo $USERNAME
$ echo $HOME
```

2. User defined variables (UDV)

Created and maintained by user. These variables are defined by **users**. A shell script allows us to set and use our **own variables** within the script. Setting variables allows you to **temporarily store data** and use it throughout the script, making the shell script more like a real computer program

- ## How to define User defined variables (UDV)

Values are assigned to user variables using an **equal sign.** No spaces can appear between the variable, the equal sign, and the value (another trouble spot for novices). Here are a few examples of assigning values to user variables.

Syntax: variable name=value
'value' is assigned to given 'variable name' and Value must be on right side = sign.

```
var1=10
var3=testing
var4="still more testing"
```

The shell script **automatically determines the data type** used for the variable value. Variables defined within the shell script maintain their values throughout the life of the shell script but are deleted when the shell script completes.

Example:

```
Var1=10   # this is ok
 20=Var1  # Error, ?
```

To define variable called 'vech' having value Bus
Vech=Bus

to print the value of vatable type:

```
echo $n
echo \$n
echo \n= $n
```

Note:

It's important to remember that when referencing a variable value you use the **dollar sign**, but when referencing the variable to assign a value to it, you do not use the dollar sign.

Rules for Naming variable name (Both UDV and System Variable)

1. Variable name must begin with Alphanumeric character or underscore character (_), followed by one or more Alphanumeric character.

2. Don't put spaces on either side of the equal sign when assigning value to variable. e.g. In following variable declaration there will be no error.

   ```
   $ no=10
   ```

   But there will be problem for any of the following variable declaration:

   ```
   $ no =10
   $ no= 10
   $ no = 10
   ```

3. Variables are case-sensitive, just like filename in Linux. For e.g.

   ```
   $ no=10
   $ No=11
   $ NO=20
   $ nO=2
   ```

   Above all are different variable name, so to **print value 20 we have to use "echo $NO"** and not any of the following

   ```
   $ echo $no # will print 10 but not 20
   $ echo $No# will print 11 but not 20
   $ echo $nO# will print 2 but not 20
   ```

4. You can define NULL variable as follows (NULL variable is variable which has no value at the time of definition). e.g.

   ```
   $ vech=
   $ vech=""
   ```

   Try to print its value by issuing following command

   ```
   $ echo $vech
   ```

   Nothing will be shown because variable has no value i.e. NULL variable.

How to print or access value of UDV (User defined variables)

To print or access UDV use following syntax
Syntax: $variablename

Define variable vech and n as follows:

vech=Bus
n=10
echo $vech
echo $n

Caution: Do not try $ **echo vech**, as it will print vech instead its value 'Bus' and $ echo n, as it will print n instead its value '10', You must use $ followed by variable name.

Open editor and write the following code.

```
myname=Ahmad
myos = OS_Linux
myno=5
echo "My name is $myname"
echo "My os is $myos"
echo "My number is myno, can you see this number"
```

Shell Arithmetic

Use to perform arithmetic operations.
Syntax: expr op1 math-operator op2 ,
expr is keyword

Examples:
expr 1 + 3
expr $var1 + $var2

Note:
expr 20 %3 - Remainder read as 20 mod 3 and remainder is 2.
expr 10 * 3 - **Multiplication use * and not * since its wild card.**

For the last statement not the following points

```
echo `expr 6 + 3`
```

i. In above command, before expr keyword we used ` **(back quote)** sign not the (**single quote i.e.** ') sign. Back quote is generally found on the key under tilde (~) on PC keyboard OR to the above of TAB key.

ii. Second, expr is also end with ` i.e. back quote.

iii. Here expr 6 + 3 is evaluated to 9, then echo command prints 9 as sum.

iv. Here if you use double quote or single quote, it will NOT work.

Example:

$ echo "expr 6 + 3"
It will print expr 6 + 3

$ echo 'expr 6 + 3'
It will print expr 6 + 3

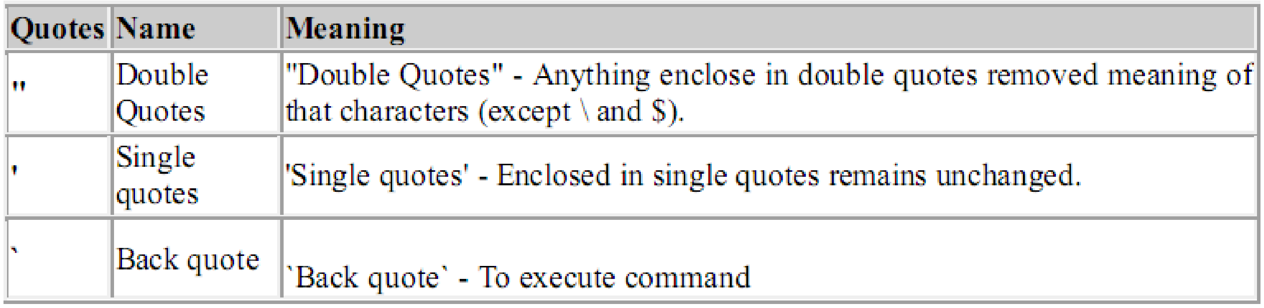
echo `expr 6 + 3`
It will print 9

$ echo "Today is date"
Can't print message with today's date.

$ echo "Today is `date`".
It will print today's date as, Today is Tue Jan ....,Can you see that the `date` statement uses back quote?

- ## More about Quotes

There are three types of quotes



Quotes	Name	Meaning
"	Double Quotes	"Double Quotes" - Anything enclose in double quotes removed meaning of that characters (except \ and $).
'	Single quotes	'Single quotes' - Enclosed in single quotes remains unchanged.
`	Back quote	`Back quote` - To execute command

- ## Exit Status

By default in Linux if particular command/shell script is executed, it return two type of values which is used to see whether command or shell script executed is successful or not.

(1) If return value is zero (0), command is successful.
(2) If return value is nonzero, command is not successful or some sort of error executing command/shell script.

But how to find out exit status of command or shell script?
Simple, to determine this exit Status you can use **echo $?** special variable of shell.

Example:
This example assumes that unknow1file does not exist on your hard drive

```
$ rm unknow1file
```

It will show error as follows

```
rm: cannot remove `unkowm1file': No such file or directory
```

and after that if you give command

```
$ echo $?
```

it will print nonzero value to indicate error.

Now give command

```
$ ls
$ echo $?
```

It will print 0 to indicate command is successful.

Lab Tasks

Task 1:

Create file name **students.txt, pstudent.txt, fstudents**. Enter students' names in **pstudent.txt** and **fstudent.txt**. Now create directory having name **UET** and copy all files to this directory. Now append the file **students.txt** with first five sorted names from **pstudent.txt** and last five sorted names from **fstudent.txt**. Then show the contents of sorted names from file **students.txt**. Change the permissions of file **students.txt** read only and both other files read and execute only.

Task 2:

1. How to define variable x with value 10 and print it on screen with **Variable name** and **Value**?
2. How to define variable xn with value Ali and print with **Variable name** and **Value**?
3. How to define two variable x=20, y=5 and then to print multiplication result of x and y with multiplication expression?

Result should be:

```
X=20
Y=5
$X *$Y = 100
20 * 5 = 100
```

4. Modify above and store division of x and y to variable called z.

Task 3:

Try the following commands and note down the exit status:

```
$ expr 1 + 3
$ echo Welcome
$ wildwest canwork?
$ date
$ echo n
```