جامعة القاهرة الجديدة التكنولوجية

Fall 2023

(5)

Linux Essentials

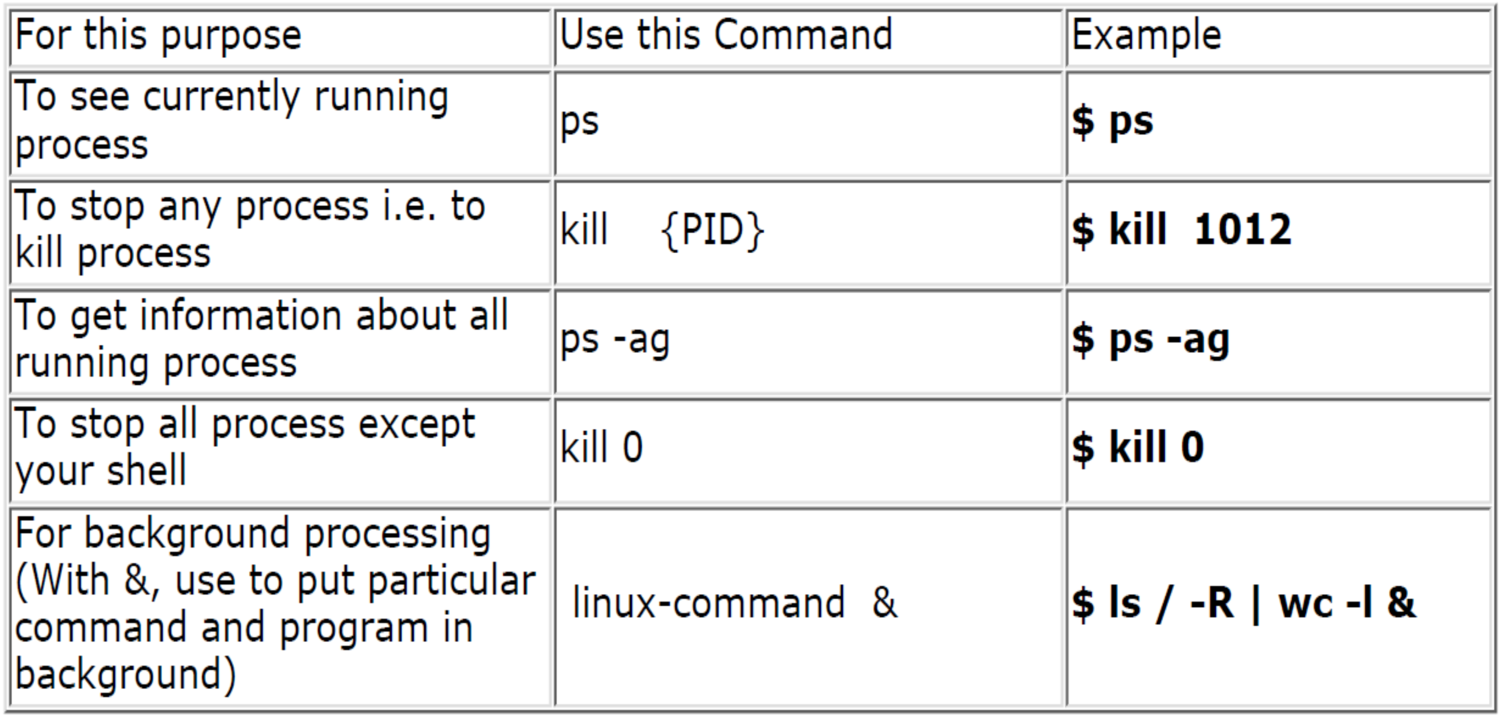
Dr. Hatem Yousry

Agenda

- Shell Variables.
- Shell Scripts.
- Command Line Processing.
- Command Line arguments.
- Exit Status.
- Shell Programming.
- if-then-fi for decision making.

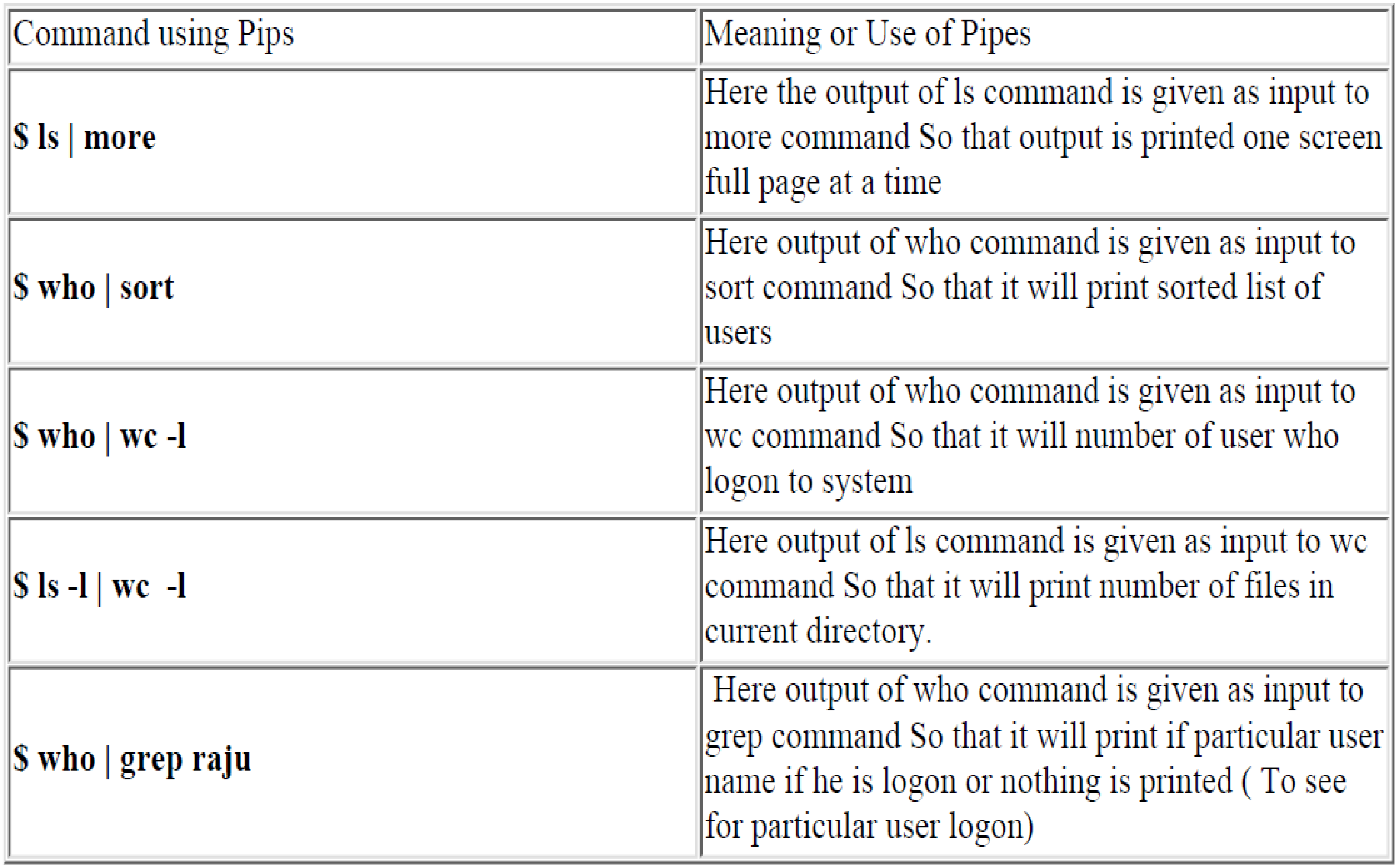
Linux Command Related with Process

- NOTE that you can only kill process which are created by yourself. A Administrator can almost kill 95-98% process. But some process can not be killed, such as **VDU Process**., A VDU is a machine with a screen which is used to display information from a computer. VDU is an abbreviation for '**visual display unit**'.



For this purpose	Use this Command	Example
To see currently running process	ps	**$ ps**
To stop any process i.e. to kill process	kill {PID}	**$ kill 1012**
To get information about all running process	ps -ag	**$ ps -ag**
To stop all process except your shell	kill 0	**$ kill 0**
For background processing (With &, use to put particular command and program in background)	linux-command &	**$ ls / -R | wc -l &**

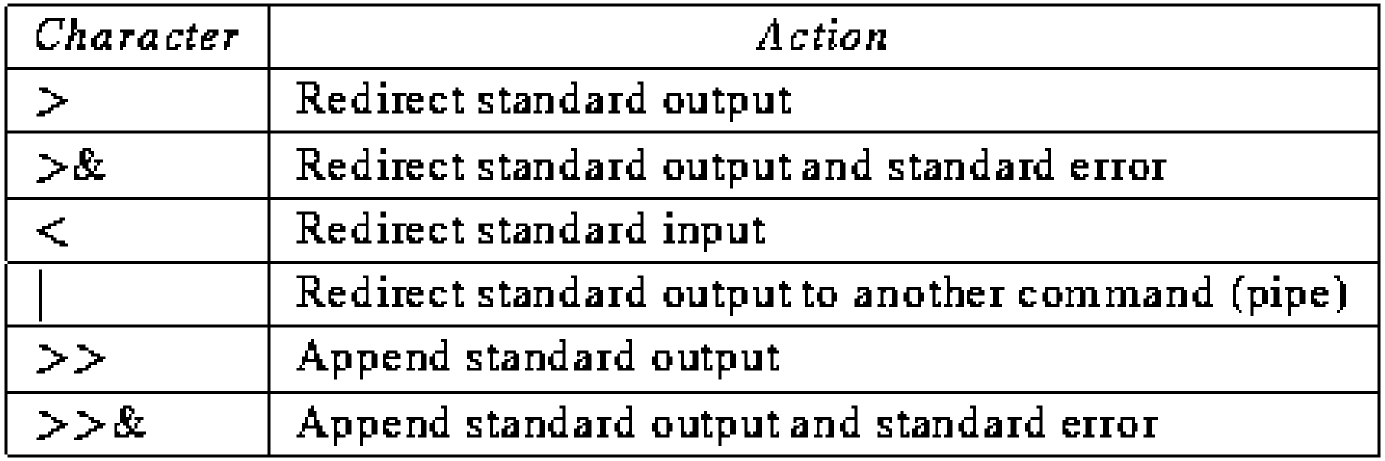
Pipes



Command using Pips	Meaning or Use of Pipes
$ ls | more	Here the output of ls command is given as input to more command So that output is printed one screen full page at a time
$ who | sort	Here output of who command is given as input to sort command So that it will print sorted list of users
$ who | wc -l	Here output of who command is given as input to wc command So that it will number of user who logon to system
$ ls -l | wc -l	Here output of ls command is given as input to wc command So that it will print number of files in current directory.
$ who | grep raju	Here output of who command is given as input to grep command So that it will print if particular user name if he is logon or nothing is printed (To see for particular user logon)

Redirection

- **Redirection of Standard output/input or Input – Output redirection.**
- Mostly all command gives output on screen or take input from keyboard, but in Linux it's possible to **send output to file or to read input from file.**
- **There are three main redirection symbols** >,>>,<



Character	*Action*
>	Redirect standard output
>&	Redirect standard output and standard error
<	Redirect standard input
|	Redirect standard output to another command (pipe)
>>	Append standard output
>>&	Append standard output and standard error

Shell Prompt Example

- Code segments and script output will be displayed as monospaced text. Command-line entries will be preceded by the **Dollar sign ($).** If your prompt is different, enter the command:
- **PS1="$ " ; export PS1**
- Then your interactions should match the examples given (such as ./my-script.sh below).
- Script output (such as **"Hello World"** below) is displayed at the start of the line.
- **$ echo '#!/bin/sh' > my-script.sh**
- **$ echo 'echo Hello World' >> my-script.sh**
- **$ chmod 755 my-script.sh**
- **$ ./my-script.sh**
- **Hello World**
- **$**

How to write shell script

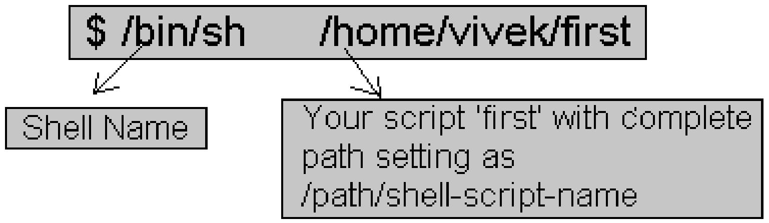
- To write shell script you can use in of the Linux's **text editor such as vi , nano or mcedit** or even you can use **cat command.** Here we are using cat command you can use any of the above text editor.
- Now we write our first script that will print "Knowledge is Power" on screen.
- First type following cat command and rest of text as its
- **$ cat > first**
- **#**
- **# My first shell script**
- **#**
- **clear**
- **echo "Knowledge is Power“**
- **Press Ctrl + D to save. Now our script is ready. To execute it type command**
- **$ ./first**
- **This will give error since we have not set Execute permission for our script first; to do this type command**
- **$ chmod +x first**
- **$ ./first**

First screen will be clear, then Knowledge is Power is printed on screen.

How to Run Shell Scripts

- Because of security of files, in Linux, the creator of Shell Script does not get execution permission by default. So if we wish to run shell script we have to do two things as follows
- **(1) Use chmod command as follows to give execution permission to our script**
- **Syntax: chmod +x shell-script-name**
- **OR Syntax: chmod 777 shell-script-name**
- **(2) Run our script as**
- **Syntax: ./your-shell-program-name**
- **For e.g.**
- **$ ./first**
- Here **'.'(dot)** is command, and used in conjunction with shell script. The dot(.) indicates to current shell that the command following the dot(.) has to be executed in the same shell i.e. without the loading of another shell in memory.

Run Shell Scripts


$ /bin/sh /home/vivek/first
Shell Name
Your script 'first' with complete path setting as /path/shell-script-name

- Or you can also try following syntax to run Shell Script
- Syntax: **bash &nbsh;&nbsh;** your-shell-program-name
- **OR /bin/sh &nbsh;&nbsh;** your-shell-program-name
- For e.g.
- **$ bash first**
- **$ /bin/sh first**
- Note that to run script, you need to have in same directory where you created your script, **if you are in different directory your script will not run (because of path settings),** For eg. Your home directory is (use **$ pwd** to see current working directory) /home/vivek. Then you created one script called 'first', after creation of this script you moved to some **other directory** lets say /home/vivek/Letters/Personal, Now if you try to execute your script it will not run, since script 'first‘ is in /home/vivek directory, to **Overcome this problem, specify complete path** of your script when ever you want to run it from other directories like giving following command
- **$ /bin/sh /home/vivek/first**

Run Shell Scripts

- Now every time you have to give all this detailed as you work in other directory, this take time and you have to remember complete path.
- **There is another way,** if you notice that all of our programs (in form of executable files) are marked as executable and can be directly executed from prompt from any directory (To see executables of our normal program give command **$ ls -l /bin or ls –l/usr/bin)** by typing command like
- **$ bc #** bc command is **used for command line calculator**. It is similar to basic calculator by using which we can do basic mathematical calculations.
- **$ cc myprg.c #** cc command is **stands for C Compiler**, It is used to compile the C language codes and create executables.
- **$ cal #** cal command is a calendar command in Linux which is **used to see the calendar of a specific month or a whole year**.

Run Shell Scripts

- **$ cal //** cal command is a calendar command in Linux which is **used to see the calendar of a specific month or a whole year**.

Option	Description
-w	Print the number of the week under each week column.
-C	Behave as if you ran cal, using its options and display output.
-M	Display weeks with Monday as the first day.
-S	Display weeks with Sunday as the first day. This is the default.

Commands Related with Shell Programming - *echo*

- ***echo [options] [string, variables...]***
- Displays text or variables value on screen.
- **Options**
- -n Do not output the trailing new line.
- -e Enable interpretation of the following backslash escaped characters in the strings:
- \a alert (bell)
- \b backspace
- \c suppress trailing new line
- \n new line
- \r carriage return
- \t horizontal tab
- \\ backslash
- **For eg. $ echo -e "An apple a day keeps away \a\t\tdoctor\n"**

More about Quotes

- **There are three types of quotes**
- **" i.e. Double Quotes**
- **' i.e. Single quotes**
- **` i.e. Back quote**
- **1."Double Quotes" - Anything enclose in double quotes removed <u>meaning</u> of that characters (except \ and $).**
- **2. 'Single quotes' - Enclosed in single quotes remains <u>unchanged.</u>**
- **3. `Back quote` - To <u>execute</u> command.**
- For e.g.
- **$ echo "Today is date"**
- Can't print message with today's date.
- **$ echo "Today is `date`".**
- Now it will print today's date as, **<u>Today is Tue Jan </u>**....,See the `date` statement uses back quote

Difference between single and double quotes in Bash

- Single quotes won't interpolate anything, but double quotes will. For example: variables, backticks, certain \ escapes, etc.
- **Example:**
- $ echo "$(echo "upg")"
- upg ← **meaning**
- $ echo '$(echo "upg")'
- $(echo "upg") ← **unchanged**
- $ echo 'expr 5 + 2'
- 7 ← **execute**

Shell Arithmetic

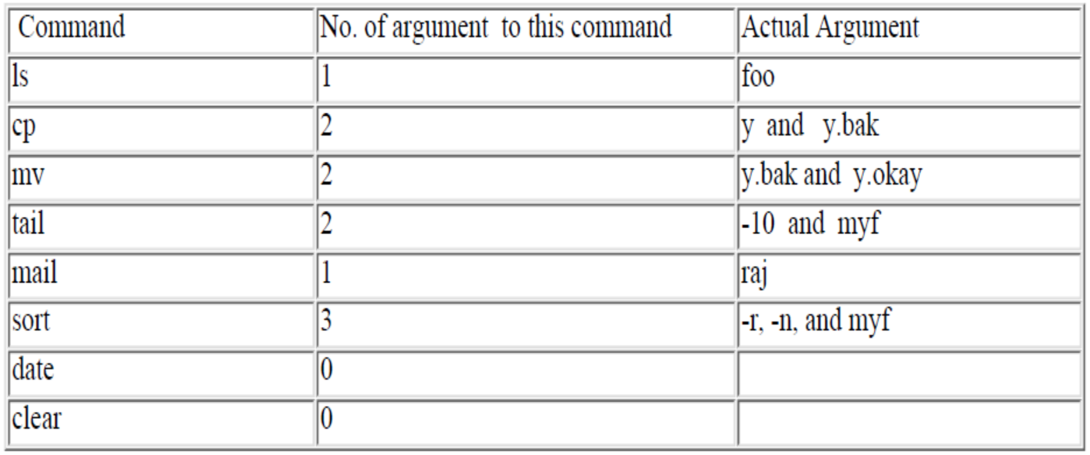
- **Use to perform arithmetic operations For e.g.**
- **$ expr 1 + 3**
- **$ expr 2 - 1**
- **$ expr 10 / 2**
- **$ expr 20 % 3 # remainder read as 20 mod 3 and remainder is 2)**
- **$ expr 10 * 3 # Multiplication use * not * since its wild card)**
- **$ echo `expr 6 + 3`**
- For the last statement not the following points
- 1) First, 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 keyboards OR To the above of TAB key.
- 2) Second, expr is also end with ` i.e. back quote.
- 3) Here expr 6 + 3 is evaluated to **9**, then echo command prints 9 as sum
- 4) Here if you use double quote or single quote, it will NOT work.
- For e.g.
- **$ echo "expr 6 + 3"** # It will print expr 6 + 3
- **$ echo 'expr 6 + 3'** # It will print expr 9

Command Line Processing

- Now try following command (assumes that **the file "grate_stories_of"** is not exist on your disk)
- **$ ls grate_stories_of**
- It will print message something like **- grate_stories_of: No such file or directory**
- Well as it turns out **ls** was the name of an actual command and shell executed this command when given the command.
- **Now it creates one question What are commands? What happened when you type $ ls grate_stories_of?** The first word on command line, **ls, is name of the command to be executed.** Everything else on command line is taken as **arguments to this command.**

Command Line Processing

- **For e.g. (Exercise)**
- **$ tail +10 myf**
- Here the name of command is **tail,** and the arguments are **+10** and myf.
- Now try to determine command and arguments from following commands:
- **$ ls foo**
- **$ cp y y.bak**
- **$ mv y.bak y.okay**
- **$ tail -10 myf**
- **$ mail raj**
- **$ sort -r -n myf**
- **$ date**
- **$ clear**



Command	No. of argument to this command	Actual Argument
ls	1	foo
cp	2	y and y.bak
mv	2	y.bak and y.okay
tail	2	-10 and myf
mail	1	raj
sort	3	-r, -n, and myf
date	0	
clear	0	

How do I use Foo in Linux?

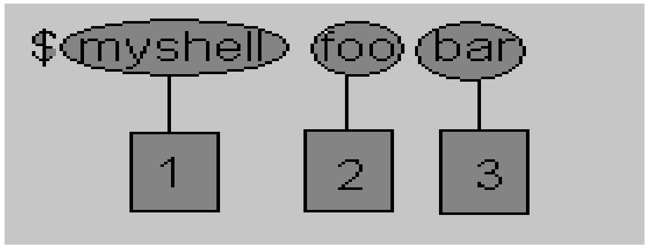
1. **more foo.txt :** displays the contents of the file named "foo.txt"
2. **rm foo. txt :** removes the file named "foo. txt". NOTE: this permanently removes the file, so use with care!
3. **cp foo. txt foo1. txt :** copies the file "foo. txt" into the file "foo1. ...
4. **mv foo. txt foo2. txt :** moves the file "foo. txt" into the file "foo2.

Why Command Line arguments required

- Let's take **rm command**, which is used to remove file, But which file you want to remove and how you will you tail this to rm command (Even rm command does not ask you name of file that would like to remove).
- So what we do is we write as command as follows
- **$ rm {file-name}**
- Here rm is command and file-name is file which you would like to remove. This way you tail to rm command which file you would like to remove. So we are doing one way communication with our command by specifying file-name.
- Also you can pass command line arguments to your script to make it more users friendly. But how we address or access command line argument in our script.
- Lets take ls command
- **$ ls -a /* #** The (ls -a) command **will enlist the whole list of the current directory including the hidden files**. When you type a command like ls a* , the shell finds all filenames in the current directory starting with a and passes them to the ls command. It means that **the file is executable**.

Command Line arguments number ($#)

- This command has 2 command line argument -a and /* is another. For shell script,
- **$ myshell foo bar**



1. Shell Script name i.e. myshell
2. First command line argument passed to myshell i.e. foo
3. Second command line argument passed to myshell i.e. bar

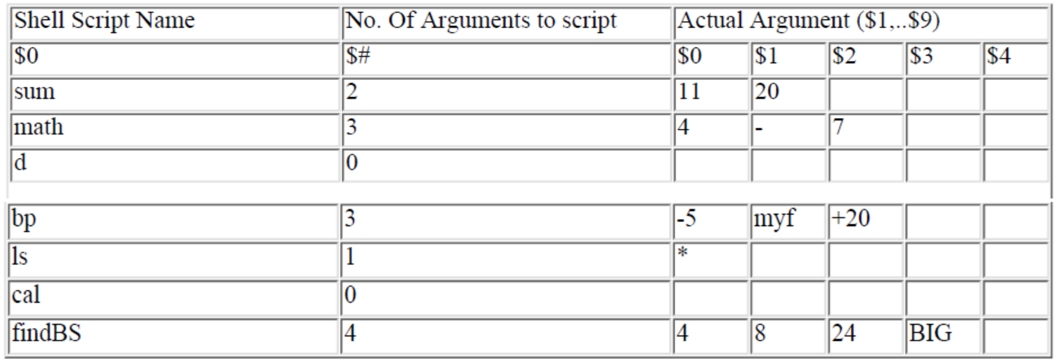
In shell if we wish to refer this command line argument we refer above as follows

1. myshell it is $0
2. foo it is $1
3. bar it is $2

Here $# will be 2 (Since foo and bar only two Arguments), $# is **the number of arguments**,

Command Line arguments number ($#)

- Please note at a time such 9 arguments can be used from $0..$9, You can also refer all of them by using **$#** (which expand to `$0,$1,$2...$9`).
- Now try to write following for commands, Shell Script Name ($0), No. of Arguments (i.e. $#), And actual argument (i.e. $1,$2 etc).
- **$ sum 11 20**
- **$ math 4 - 7**
- **$ d**
- **$ bp -5 myf +20**
- **$ ls ***
- **$ cal**
- **$ findBS 4 8 24 BIG**



Shell Script Name	No. Of Arguments to script	Actual Argument ($1,..$9)				
$0	$#	$0	$1	$2	$3	$4
sum	2	11	20			
math	3	4	-	7		
d	0					
bp	3	-5	myf	+20		
ls	1	*				
cal	0					
findBS	4	4	8	24	BIG	

Command Line arguments number ($#)

- For e.g. now will write script to print command ling argument and we will see how to access them
- **$ cat > demo**
- **#!/bin/sh**
- **#**
- **# Script that demos, command line args**
- **#**
- **echo "Total number of command line argument are $#"**
- **echo "$0 is script name"**
- **echo "$1 is first argument"**
- **echo $2 is second argument"**
- **echo "All of them are :- $*"**
- Save the above script by pressing ctrl+d, now make it executable
- **$ chmod +x demo**
- **$ ./demo Hello World**
- **$ cp demo ~/bin**
- **$ demo**

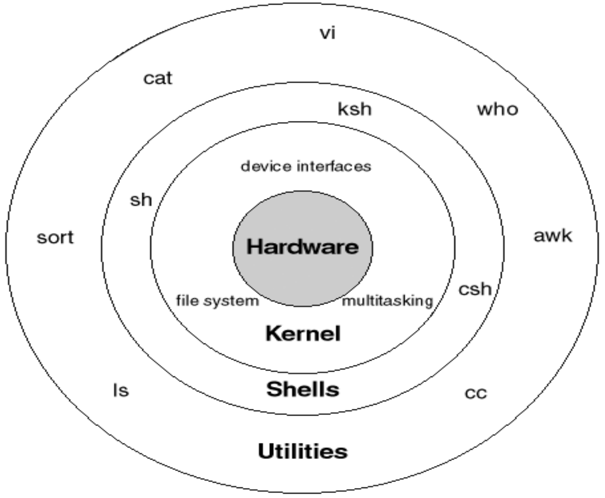
Exit Status

- By default in Linux if particular command is executed, it return two type of values, (Values are used to see whether command is successful or not) if return value is zero (0), command is successful, if return value is nonzero (>0), command is not successful or some sort of error executing command/shell script.
- **This value is know as Exit Status of that command.** To determine this exit Status we use $? variable of shell. For eg.
- **$ 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(>0) to indicate error.**

Exit Status

- Now give command
- **$ ls**
- **$ echo $?**
- It will print 0 to indicate command is successful. Try the following commands and not down there exit status
- **$ expr 1 + 3**
- **$ echo $?**
- **$ echo Welcome**
- **$ echo $?**
- **$ wildwest canwork?**
- **$ echo $?**
- **$ date**
- **$ echo $?**
- **$ echon $?**
- **$ echo $?**

Shell Programming



- Using a shell script is most useful for **repetitive tasks that may be time consuming to execute by typing one line at a time**. A few examples of applications shell scripts can be used for include: Automating the code compiling process. Running a program or creating a program environment.
- A Unix shell is **both a command interpreter and a programming language**. As a command interpreter, the shell provides the user interface to the rich set of GNU utilities. The programming language features allow these utilities to be combined. ... Shells may be used interactively or non-interactively.

if-then-fi for decision making

- ***if-then-fi for decision making*** *is shell script* Before making any decision in Shell script you must know following things Type **bc** at $ prompt to start Linux calculator program
- **$ bc**
- After this command bc is started and waiting for you commands, i.e. give it some calculation as follows type 5 + 2 as
- **5 + 2**
- **7**
- 7 is response of bc i.e. addition of 5 + 2 you can even try
- **5 - 2**
- **5 / 2**
- Now what happened if you type 5 > 2 as follows
- **5 > 2**
- **0**
- 0 (Zero) is response of bc,

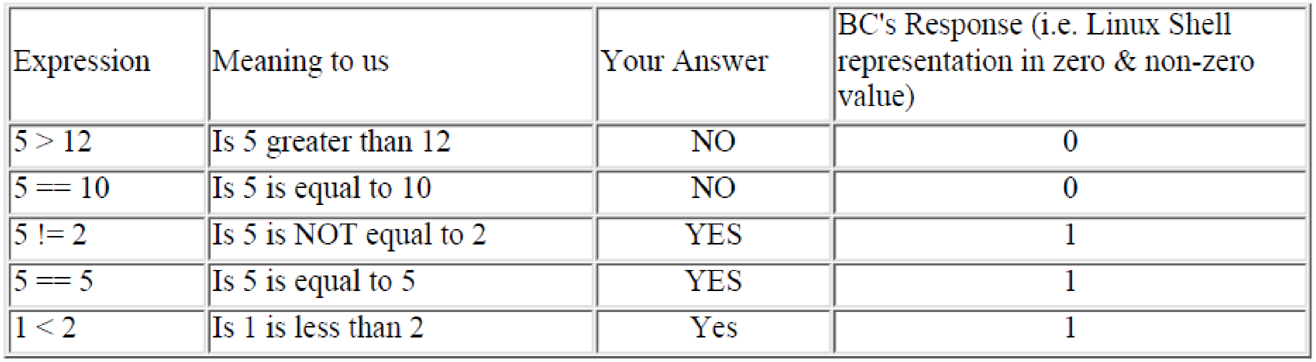
bc # bc command is **used for command line calculator**. It is similar to basic calculator by using which we can do basic mathematical calculations.

if-then-fi for decision making

- How? Here it compare 5 with 2 as, Is 5 is greater then 2, (If I ask same question to you, your answer will be YES) In Linux (**bc**) gives this 'YES' answer by showing 1 (One) value.
- It means when ever there is any type of comparison in Linux Shell It gives only two answer **one is YES** and Zero for NO is other.

if-then-fi for decision making

- **Try following in bc to clear your Idea and not down bc's response**
- **5 > 12**
- **5 == 10**
- **5 != 2**
- **5 == 5**
- **12 < 2**



Expression	Meaning to us	Your Answer	BC's Response (i.e. Linux Shell representation in zero & non-zero value)
5 > 12	Is 5 greater than 12	NO	0
5 == 10	Is 5 is equal to 10	NO	0
5 != 2	Is 5 is NOT equal to 2	YES	1
5 == 5	Is 5 is equal to 5	YES	1
1 < 2	Is 1 is less than 2	Yes	1

- Now will see, if condition which is used for decision making in shell script, If given condition is true then command1 is executed.

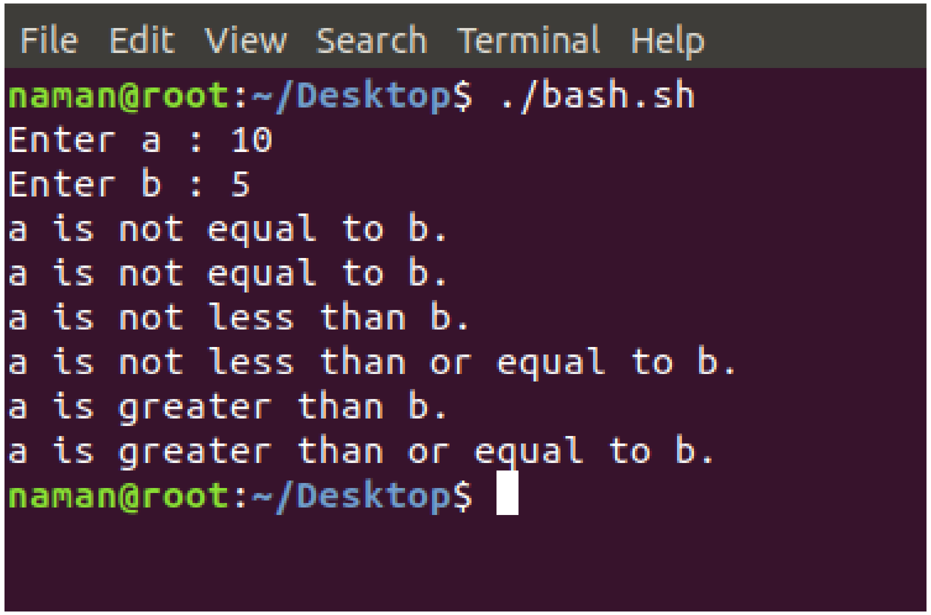
if-then-fi for decision making

- **Syntax:**
- ***if condition***
- ***then***
- ***command1 if condition is true or if exit status***
- ***of condition is 0 (zero)***
- **...**
- **...**
- ***Fi***
- Here condition is nothing but comparison between two values, for compression we can use test or [expr] statements or even exist status can be also used. An expression is nothing but combination of values, relational operator (such as >,<, <> etc) and mathematical operators (such as +, -, / etc).

```
#!/bin/bash

#reading data from the user
read -p 'Enter a : ' a
read -p 'Enter b : ' b

if(( $a==$b ))
then
        echo a is equal to b.
else
        echo a is not equal to b.
fi
if(( $a!=$b ))
then
        echo a is not equal to b.
else
        echo a is equal to b.
fi
if(( $a<$b ))
then
        echo a is less than b.
else
        echo a is not less than b.
fi
if(( $a<=$b ))
then
        echo a is less than or equal to b.
else
        echo a is not less than or equal to b.
fi
if(( $a>$b ))
then
        echo a is greater than b.
else
        echo a is not greater than b.
fi
if(( $a>=$b ))
then
        echo a is greater than or equal to b.
else
        echo a is not greater than or equal to b.
fi
```



```
naman@root:~/Desktop$ ./bash.sh
Enter a : 10
Enter b : 5
a is not equal to b.
a is not equal to b.
a is not less than b.
a is not less than or equal to b.
a is greater than b.
a is greater than or equal to b.
naman@root:~/Desktop$
```

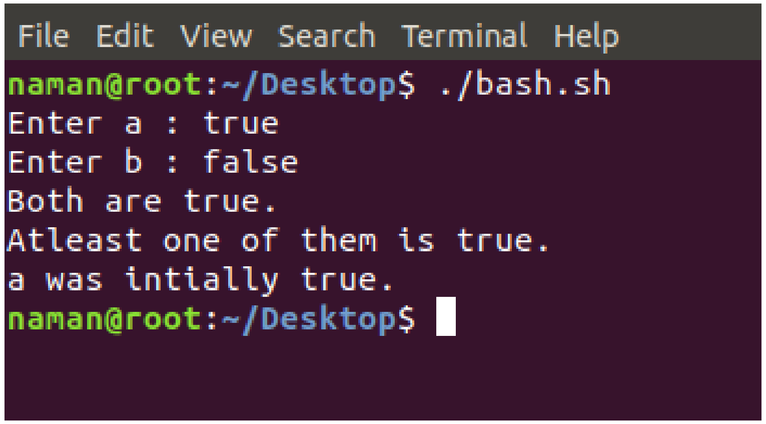
```
#!/bin/bash

#reading data from the user
read -p 'Enter a : ' a
read -p 'Enter b : ' b

if(($a == "true" & $b == "true" ))
then
        echo Both are true.
else
        echo Both are not true.
fi

if(($a == "true" || $b == "true" ))
then
        echo Atleast one of them is true.
else
        echo None of them is true.
fi

if(( ! $a == "true" ))
then
        echo "a" was initially false.
else
        echo "a" was initially true.
fi
```



```
naman@root:~/Desktop$ ./bash.sh
Enter a : true
Enter b : false
Both are true.
Atleast one of them is true.
a was intially true.
naman@root:~/Desktop$
```

if-then-fi for decision making

- **Following are all examples of expression:**
- 5 > 2
- 3 + 6
- 3 * 65
- a < b
- c > 5
- c > 5 + 30 -1
- **Type following command (assumes you have file called foo)**
- **$ cat foo**
- **$ echo $?**
- The cat command return zero(0) on successful, this can be used in if condition as follows, Write shell script as
- **$ cat > showfile**
- **#!/bin/sh**
- **#**
- **#Script to print file**
- **#**
- **if cat $1**
- **then**
- **echo -e "\n\nFile $1, found and successfully echoed"**
- **fi**

if-then-fi for decision making

- Now run it.
- **$ chmod +x showfile**
- **$./showfile foo**
- Here
- **$ ./showfile foo**
- Our shell script name is showfile($0) and foo is argument (which is $1).Now we compare as follows if cat $1 (i.e. if cat foo).

if-then-fi for decision making

- Now if cat command finds foo file and if its successfully shown on screen, it means our cat command is successful and its exist status is 0 (indicates success) So our if condition is also true and hence statement echo -e "\n\nFile $1, found and successfully echoed" is proceed by shell.
- Now if cat command is not successful then it returns non-zero value (indicates some sort of failure) and this statement echo -e "\n\nFile $1, found and successfully echoed" is skipped by our shell.
- Now try to write answer for following
- **1) Create following script**
- **cat > trmif**
- **#**
- **# Script to test rm command and exist status**
- **#**
- **if rm $1**
- **then**
- **echo "$1 file deleted"**
- **fi**
- **(Press Ctrl + d to save)**
- **$ chmod +x trmif**

if-then-fi for decision making

- **Now answer the following**
- A) There is file called foo, on your disk and you give command, **$ ./trmfi foo** what will be output.
- B) If bar file not present on your disk and you give command, $ **./trmfi bar** what will be output.
- C) And if you type **$ ./trmfi**, What will be output.

Thank You

Linux

Dr. Hatem Yousry
E-mail: Hyousry@nctu.edu.eg