**How to write shell script**

Following steps are required to write shell script:

(1) Use any editor like vi or mcedit to write shell script.

(2) After writing shell script set execute permission for your script as follows  
*syntax:*  
chmod permission your-script-name

*Examples:*  
$ chmod +x your-script-name  
$ chmod 755 your-script-name

***Note:*** This will set read write execute(7) permission for owner, for group and other permission is read and execute only(5).

(3) Execute your script as  
*syntax:*  
bash your-script-name  
sh your-script-name  
./your-script-name

*Examples:*  
$ bash bar  
$ sh bar  
$ ./bar

***NOTE*** In the last syntax ./ means current directory, But only . (dot) means execute given command file in current shell without starting the new copy of shell, The syntax for . (dot) command is as follows  
*Syntax:*  
. command-name

*Example:*  
$ . foo

Now you are ready to write first shell script that will print "Knowledge is Power" on screen. See the [common vi command list](http://www.freeos.com/guides/lsst/misc.htm#commonvi) , if you are new to vi.

|  |
| --- |
| $ vi first # # My first shell script # clear echo "Knowledge is Power" |

After saving the above script, you can run the script as follows:  
$ ./first

This will not run script since we have not set execute permission for our script *first*; to do this type command  
$ chmod 755 first  
$ ./first

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

|  |  |
| --- | --- |
| **Script Command(s)** | **Meaning** |
| $ vi first | Start vi editor |
| # # My first shell script # | # followed by any text is considered as comment. Comment gives more information about script, logical explanation about shell script. *Syntax:* # comment-text |
| clear | clear the screen |
| 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" |

***Exercise:***  
1)Write following shell script, save it, execute it and note down the it's output.

|  |
| --- |
| $ vi ginfo # # # Script to print user information who currently login , current date & time # clear echo "Hello $USER" echo "Today is \c ";date echo "Number of user login : \c" ; who | wc -l echo "Calendar" cal exit 0 |

**Variables in Shell**

To process our data/information, data must be kept in computers RAM memory. RAM memory is divided into small locations, and each location had unique number called memory location/address, which is used to hold our data. Programmer can give a unique name to this memory location/address called memory variable or variable (Its a named storage location that may take different values, but only one at a time).

In Linux (Shell), there are two types of variable:  
(1) **System variables** - Created and maintained by Linux itself. This type of variable defined in CAPITAL LETTERS.  
(2) **User defined variables (UDV)** - Created and maintained by user. This type of variable defined in lower letters.

You can see system variables by giving command like **$ set**, some of the important System variables are:

|  |  |
| --- | --- |
| **System Variable** | **Meaning** |
| BASH=/bin/bash | Our shell name |
| BASH\_VERSION=1.14.7(1) | Our shell version name |
| COLUMNS=80 | No. of columns for our screen |
| HOME=/home/vivek | Our home directory |
| LINES=25 | No. of columns for our screen |
| LOGNAME=students | students Our logging name |
| OSTYPE=Linux | Our Os type |
| PATH=/usr/bin:/sbin:/bin:/usr/sbin | Our path settings |
| PS1=[\u@\h \W]\$ | Our prompt settings |
| PWD=/home/students/Common | Our current working directory |
| SHELL=/bin/bash | Our shell name |
| USERNAME=vivek | User name who is currently login to this PC |

***NOTE*** that Some of the above settings can be different in your PC/Linux environment. You can print any of the above variables contains as follows:  
$ echo $USERNAME  
$ echo $HOME

Exercise:  
1) If you want to print your home directory location then you give command:  
a)$ echo $HOME

**OR**

(b)$ echo HOME

# How to define User defined variables (UDV)

To define UDV use following syntax  
*Syntax:*  
variable name=value

'**value**' is assigned to given '**variable name**' and Value must be on right side = sign.  
  
*Example:*$ no=10# this is ok  
$ 10=no# Error, NOT Ok, Value must be on right side of = sign.  
To define variable called 'vech' having value Bus  
$ vech=Bus  
To define variable called n having value 10  
$ n=10

# 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. For e.g. Valid shell variable are as follows  
**HOME  
SYSTEM\_VERSION  
vech  
no**

(2) Don't put spaces on either side of the equal sign when assigning value to variable. For 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) For e.g.  
$ vech=  
$ vech=""  
Try to print it's value by issuing following command  
$ echo $vech  
Nothing will be shown because variable has no value i.e. NULL variable.

(5) Do not use **?,\***etc, to name your variable names.

**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  
To print contains of variable 'vech' type  
$ echo $vech  
It will print 'Bus',To print contains of variable 'n' type command as follows  
$ 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*.

**Exercise**  
Q.1.How to Define variable x with value 10 and print it on screen.  
Q.2.How to Define variable xn with value Rani and print it on screen  
Q.3.How to print sum of two numbers, let's say 6 and 3?  
Q.4.How to define two variable x=20, y=5 and then to print division of x and y (i.e. x/y)  
Q.5.Modify above and store division of x and y to variable called z  
Q.6.Point out error if any in following script

|  |
| --- |
| $ vi variscript # # # Script to test MY knowledge about variables! # myname=Vivek myos = TroubleOS myno=5 echo "My name is $myname" echo "My os is $myos" echo "My number is myno, can you see this number" |

# echo Command

Use echo command to display text or value of variable.

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 e.g. **$ echo -e "An apple a day keeps away \a\t\tdoctor\n"**

# Shell Arithmetic

Use to perform arithmetic operations.

*Syntax:*  
expr op1 math-operator op2  
  
*Examples:*  
$ expr 1 + 3  
$ expr 2 - 1  
$ expr 10 / 2  
$ expr 20 % 3  
$ expr 10 \\* 3  
$ echo `expr 6 + 3`

**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

(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 keyboard 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 6 + 3

**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 |

*Example*:  
**$ 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?

# 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.

This value is know as ***Exit Status***.

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

For e.g. (This example assumes that **unknow1file** doest 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.

Exercise  
Try the following commands and not down the exit status:  
$ expr 1 + 3  
$ echo $?

$ echo Welcome  
$ echo $?

$ wildwest canwork?  
$ echo $?

$ date  
$ echo $?

$ echon $?  
$ echo $?

**The read Statement**

Use to get input (data from user) from keyboard and store (data) to variable.  
*Syntax:*  
read variable1, variable2,...variableN

Following script first ask user, name and then waits to enter name from the user via keyboard. Then user enters name from keyboard (after giving name you have to press ENTER key) and entered name through keyboard is stored (assigned) to variable fname.

|  |
| --- |
| $ vi sayH # #Script to read your name from key-board # echo "Your first name please:" read fname echo "Hello $fname, Lets be friend!" |

Run it as follows:  
$ chmod 755 sayH  
$ ./sayH  
*Your first name please:****vivek*** *Hello vivek, Lets be friend!*

**Wild cards (Filename Shorthand or meta Characters)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Wild card /Shorthand** | **Meaning** | **Examples** | |
| **\*** | Matches any string or group of characters. | **$ ls \*** | will show all files |
| **$ ls a\*** | will show all files whose first name is starting with letter 'a' |
| **$ ls \*.c** | will show all files having extension .c |
| **$ ls ut\*.c** | will show all files having extension .c but file name must begin with 'ut'. |
| **?** | Matches any single character. | **$ ls ?** | will show all files whose names are 1 character long |
| **$ ls fo?** | will show all files whose names are 3 character long and file name begin with fo |
| **[...]** | Matches any one of the enclosed characters | **$ ls [abc]\*** | will show all files beginning with letters a,b,c |

**Note:**  
[..-..] A pair of characters separated by a minus sign denotes a range.

*Example*:  
**$ ls /bin/[a-c]\***

Will show all files name beginning with letter a,b or c like

   /bin/arch           /bin/awk           /bin/bsh     /bin/chmod           /bin/cp  
   /bin/ash           /bin/basename   /bin/cat      /bin/chown           /bin/cpio  
   /bin/ash.static   /bin/bash          /bin/chgrp   /bin/consolechars  /bin/csh

But  
**$ ls /bin/[!a-o]  
$ ls /bin/[^a-o]**

If the first character following the [ is a ! or a ^ ,then any character not enclosed is matched i.e. do not show us file name that beginning with a,b,c,e...o, like

   /bin/ps            /bin/rvi              /bin/sleep /bin/touch      /bin/view  
   /bin/pwd           /bin/rview        /bin/sort   /bin/true        /bin/wcomp  
   /bin/red           /bin/sayHello     /bin/stty   /bin/umount   /bin/xconf  
   /bin/remadmin  /bin/sed           /bin/su      /bin/uname     /bin/ypdomainname  
   /bin/rm            /bin/setserial    /bin/sync   /bin/userconf  /bin/zcat  
   /bin/rmdir         /bin/sfxload      /bin/tar    /bin/usleep  
   /bin/rpm           /bin/sh            /bin/tcsh    /bin/vi

# More command on one command line

*Syntax:*  
command1;command2  
To run two command with one command line.

*Examples:*  
**$ date;who**  
Will print today's date followed by users who are currently login. Note that You can't use  
**$ date who**  
for same purpose, you must put semicolon in between date and who command.

**Command Line Processing**

Try the following command (assumes that the file "**grate\_stories\_of**" is not exist on your system)  
**$ ls grate\_stories\_of**It will print message something like - *grate\_stories\_of: No such file or directory*.

**ls** is the name of an *actual command* and shell executed this command when you type command at shell prompt. Now it creates one more question **What are commands?** What happened when you type *$ ls grate\_stories\_of*?

The first word on command line is,**ls** - is name of the command to be executed.  
Everything else on command line is taken *as arguments to this command*. For e.g.  
**$ tail +10 myf**  
Name of command is **tail**, and the arguments are **+10**and **myf**.

**Exercise**  
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

Answer:

|  |  |  |
| --- | --- | --- |
| **Command** | **No. of argument  to this command (i.e $#)** | **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 |  |

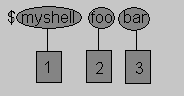
**NOTE:**  
**$#** holds number of arguments specified on command line. And**$\*** or **$@** refer to all arguments passed to script.

**Why Command Line arguments required**

1. Telling the command/utility which option to use.
2. Informing the utility/command which file or group of files to process (reading/writing of files).

Let's take rm command, which is used to remove file, but which file you want to remove and how you will tail this to rm command (even rm command don't ask you name of file that you would like to remove). So what we do is we write command as follows:  
**$ rm {file-name}**  
Here rm is command and filename is file which you would like to remove. This way you tail rm command which file you would like to remove. So we are doing one way communication with our command by specifying filename Also you can pass command line arguments to your script to make it more users friendly. But how we access command line argument in our script.

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



http://www.freeos.com/guides/lsst/images/1.gif Shell Script name i.e. myshell  
http://www.freeos.com/guides/lsst/images/2.gif First command line argument passed to myshell i.e. foo  
http://www.freeos.com/guides/lsst/images/3.gif 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

http://www.freeos.com/guides/lsst/images/1.gif myshell it is $0  
http://www.freeos.com/guides/lsst/images/2.gif foo it is $1  
http://www.freeos.com/guides/lsst/images/2.gif bar it is $2

Here **$#**(built in shell variable ) will be 2 (Since foo and bar only two Arguments), Please note at a time such 9 arguments can be used from $1..$9, You can also refer all of them by using $\* (which expand to `$1,$2...$9`). Note that $1..$9 i.e command line arguments to shell script is know as "*positional parameters*".

**Exercise**  
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

**Answer**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Shell Script Name** | **No. Of Arguments to script** | **Actual Argument ($1,..$9)** | | | | |
| ***$0*** | ***$#*** | ***$1*** | ***$2*** | ***$3*** | ***$4*** | ***$5*** |
| 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 |  |

Following script is used to print command ling argument and will show you how to access them:

|  |
| --- |
| $ vi 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 :- $\* or $@" |

Run it as follows

Set execute permission as follows:  
**$ chmod 755 demo**

Run it & test it as follows:  
**$ ./demo Hello World**

If test successful, copy script to your own bin directory (Install script for private use)  
**$ cp demo ~/bin**

Check whether it is working or not (?)  
**$ demo**  
**$ demo Hello World**

**NOTE:**After this, for any script you have to used above command, in sequence, I am not going to show you all of the above command(s) for rest of Tutorial.

Also note that you ***can't assigne the new value to command line arguments i.e positional parameters***. So following all statements in shell script are invalid:  
**$1 = 5  
$2 = "My Name"**

# Redirection of Standard output/input i.e. Input - Output redirection

Mostly all command gives output on screen or take input from keyboard, but in Linux (and in other OSs 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 put output of ls command to filename.

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

(1) > Redirector Symbol  
*Syntax:*  
Linux-command > filename  
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. For e.g. To send output of ls command give  
**$ ls > myfiles**  
Now if '**myfiles**' file exist in your current directory it will be overwritten without any type of warning.

(2) >> Redirector Symbol  
*Syntax:*  
Linux-command >> filename  
To output Linux-commands result (output of command or shell script) to END of file. Note that if file exist , it will be opened and new information/data will be written to END of file, without losing previous information/data, And if file is not exist, then new file is created. For e.g. To send output of date command to already exist file give command  
**$ date >> myfiles**

(3) < Redirector Symbol  
*Syntax:*  
Linux-command < filename  
To take input to Linux-command from file instead of key-board. For e.g. To take input for cat command give  
**$ cat < myfiles**

http://www.freeos.com/guides/lsst/images/bulb.gif [Click here to learn more about I/O Redirection](http://www.freeos.com/guides/lsst/ch04sec4.html)

You can also use above redirectors simultaneously as follows  
Create text file sname as follows

**$cat > sname**  
vivek  
ashish  
zebra  
babu  
*Press CTRL + D to save.*

Now issue following command.  
**$ sort < sname > sorted\_names**  
**$ cat sorted\_names**  
ashish  
babu  
vivek  
zebra

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

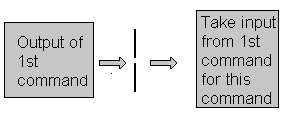
Try one more example to clear your idea:  
**$ tr "[a-z]" "[A-Z]" < sname > cap\_names**  
**$ cat cap\_names**  
VIVEK  
ASHISH  
ZEBRA  
BABU

[tr](http://www.freeos.com/guides/lsst/ch05sec05.html) command is used to translate all lower case characters to upper-case letters. It take input from sname file, and tr's output is redirected to cap\_names file.

**Future Point :** Try following command and find out most important point:  
**$ sort > new\_sorted\_names < sname**  
**$ cat new\_sorted\_names**

**Pipes**

A pipe is a way to connect the output of one program to the input of another program without any temporary file.



Pipe Defined as:  
"*A pipe is nothing but a temporary storage place where the output of one command is stored and then passed as the input for second command. Pipes are used to run more than two commands ( Multiple commands) from same command line.*"

*Syntax:*  
command1 | command2

*Examles:*

|  |  |
| --- | --- |
| **Command using Pipes** | **Meaning or Use of Pipes** |
| **$ ls | more** | 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** | Output of who command is given as input to sort command So that it will print sorted list of users |
| **$ who | sort > user\_list** | Same as above except output of sort is send to (redirected) user\_list file |
| **$ who | wc -l** | 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** | 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** | 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 particular user is logon or not) |

|  |
| --- |
|  |

# Filter

If a Linux command accepts its input from the standard input and produces its output on standard output is know as a filter. A filter performs some kind of process on the input and gives output. For e.g.. Suppose you have file called 'hotel.txt' with 100 lines data, And from 'hotel.txt' you would like to print contains from line number 20 to line number 30 and store this result to file called 'hlist' then give command:  
**$ tail +20 < hotel.txt | head -n30 >hlist**

Here **head** command is filter which takes its input from tail command (tail command start selecting from line number 20 of given file i.e. hotel.txt) and passes this lines as input to head, whose output is redirected to 'hlist' file.

Consider one more following example  
**$ sort < sname | uniq > u\_sname**

Here [uniq](http://www.freeos.com/guides/lsst/ch05sec08.html) is filter which takes its input from sort command and passes this lines as input to uniq; Then uniqs output is redirected to "u\_sname" file.