**<https://github.com/ramireddygangadhar/ashok.git>**

git remote add origin https://github.com/ramireddygangadhar/ashok.git

**What is Linux Shell ?**

Computer understand the language of 0's and 1's called binary language.

In early days of computing, instruction are provided using binary language, which is difficult for all of us, to read and write. So in Os there is special program called Shell. Shell accepts your instruction or commands in English (mostly) and if its a valid command, it is pass to kernel.

Shell is a user program or it's environment provided for user interaction. Shell is an command language interpreter that executes commands read from the standard input device (keyboard) or from a file.

Shell is not part of system kernel, but uses the system kernel to execute programs, create files etc.

Several shell available with Linux including:

|  |  |  |  |
| --- | --- | --- | --- |
| **Shell Name** | **Developed by** | **Where** | **Remark** |
| BASH ( Bourne-Again SHell ) | Brian Fox and Chet Ramey | Free Software Foundation | Most common shell in Linux. It's Freeware shell. |
| CSH (C SHell) | Bill Joy | University of California (For BSD) | The C shell's syntax and usage are very similar to the C programming language. |
| KSH (Korn SHell) | David Korn | AT & T Bell Labs | -- |
| TCSH | See the man page. Type $ man tcsh | -- | TCSH is an enhanced but completely compatible version of the Berkeley UNIX C shell (CSH). |

***Tip:*** To find all available shells in your system type following command:  
**$ cat /etc/shells**

***Note***that each shell does the same job, but each understand a different command syntax and provides different built-in functions.

In MS-DOS, Shell name is COMMAND.COM which is also used for same purpose, but it's not as powerful as our Linux Shells are!

Any of the above shell reads command from user (via Keyboard or Mouse) and tells Linux Os what users want. If we are giving commands from keyboard it is called command line interface ( Usually in-front of $ prompt, This prompt is depend upon your shell and Environment that you set or by your System Administrator, therefore you may get different prompt ).

***Tip:*** To find your current shell type following command  
**$ echo $SHELL**

**What is Shell Script ?**

Normally shells are interactive. It means shell accept command from you (via keyboard) and execute them. But if you use command one by one (sequence of 'n' number of commands) , the you can store this sequence of command to text file and tell the shell to execute this text file instead of entering the commands. This is know as ***shell script***.

Shell script defined as:"*Shell Script is****series of command****written****in plain text file****. Shell script is just like batch file is MS-DOS but have more power than the MS-DOS batch file.*"

**Why to Write Shell Script ?**

* Shell script can take input from user, file and output them on screen.
* Useful to create our own commands.
* Save lots of time.
* To automate some task of day today life.
* System Administration part can be also automated.

**Getting started with Shell Programming**

In this part of tutorial you are introduce to shell programming, how to write script, execute them etc. We will getting started with writing small shell script, that will print "Knowledge is Power" on screen. Before starting with this you should know

* How to use text editor such as vi, see the [common vi command](http://www.freeos.com/guides/lsst/misc.htm#commonvi) for more information.
* Basic command navigation
* **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" |

* http://www.freeos.com/guides/lsst/images/bulb.gif [How Shell Locates the file](http://www.freeos.com/guides/lsst/misc.htm#howshell) (My own bin directory to execute script)
* ***Tip:*** For shell script file try to give file extension such as .sh, which can be easily identified by you as shell script.
* ***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 |

* **Future Point:** At the end why statement exit 0 is used? See [exit status](http://www.freeos.com/guides/lsst/ch02sec09.html) for more information.
* **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
* Which of the above command is correct & why? [Click here for answer.](http://www.freeos.com/guides/lsst/misc.htm#echoans4doller)
* **Caution:**Do not modify System variable this can some time create problems.

# 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" |

[For Answers Click here](http://www.freeos.com/guides/lsst/misc.htm#variexceans)

After clicking the above link

**Answer to Variable sections exercise**

Q.1.How to Define variable x with value 10 and print it on screen.  
**$ x=10  
$ echo $x**  
Q.2.How to Define variable xn with value Rani and print it on screen  
For Ans. Click here  
**$ xn=Rani  
$ echo $xn**  
Q.3.How to print sum of two numbers, let's say 6 and 3  
**$ echo 6 + 3**This will print 6 + 3, not the sum 9, To do sum or math operations in shell use expr, syntax is as follows    
Syntax:*expr   op1   operator   op2*Where, op1 and op2 are any Integer Number (Number without decimal point) and operator can be  
+ Addition  
- Subtraction  
/ Division  
% Modular, to find remainder For e.g. 20 / 3 = 6 , to find remainder 20 % 3 = 2, (Remember its integer calculation)  
\\* Multiplication  
**$ expr 6 + 3**  
Now It will print sum as 9 , But  
**$ expr 6+3**  
will not work because space is required between number and operator (See Shell Arithmetic)  
  
Q.4.How to define two variable x=20, y=5 and then to print division of x and y (i.e. x/y)  
For Ans. Click here  
**$x=20  
$ y=5  
$ expr x / y**  
  
Q.5.Modify above and store division of x and y to variable called z  
For Ans. Click here  
**$ x=20  
$ y=5  
$ z=`expr x / y`  
$ echo $z**

Q.6.Point out error if any in following script

|  |
| --- |
| **$ vi   variscript** **# # # Script to test MY knolwdge about variables! # myname=Vivek myos   =  TroubleOS    -----> ERROR 1 myno=5 echo "My name is $myname" echo "My os is $myos" echo "My number is   myno,   can you see this number"  ----> ERROR 2** |

**ERROR 1**[**Read this**](http://www.freeos.com/guides/lsst/ch02sec05.html#erronotagin)

**ERROR 2**[**Read this**](http://www.freeos.com/guides/lsst/ch02sec05.html#erronotagin)

Following script should work now, after bug fix!

|  |
| --- |
| **$ vi   variscript** **# # # Script to test MY knolwdge 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"** |

[http://www.freeos.com/guides/lsst/images/backarr.gif](javascript:history.back(1))

**Parameter substitution.**

Now consider following command  
$(**$ echo 'expr 6 + 3')**

The command (**$ echo 'expr 6 + 3')**is know as **Parameter substitution.**When a command is enclosed in backquotes, the command get executed and we will get output. Mostly this is used in conjunction with other commands. For e.g.

|  |
| --- |
| **$pwd $cp /mnt/cdrom/lsoft/samba\*.rmp `pwd`** |

Now suppose we are working in directory called "/home/vivek/soft/artical/linux/lsst" and I want to copy some samba files from "/mnt/cdrom/lsoft" to my current working directory, then my command will be something like

**$cp   /mnt/cdrom/lsoft/samba\*.rmp    /home/vivek/soft/artical/linux/lsst**

Instead of giving above command I can give command as follows

**$cp  /mnt/cdrom/lsoft/samba\*.rmp  `pwd`**

Here file is copied to your working directory. See the last **Parameter substitution**of `pwd` command, expand it self to **/home/vivek/soft/artical/linux/lsst.**This will save my time. **$cp  /mnt/cdrom/lsoft/samba\*.rmp  `pwd`**

**Future Point:**What is difference between following two command? **$cp  /mnt/cdrom/lsoft/samba\*.rmp  `pwd`**A N D **$cp  /mnt/cdrom/lsoft/samba\*.rmp  .**Try to note down output of following Parameter substitution.

|  |
| --- |
| **$echo "Today date is `date`" $cal > menuchoice.temp.$$ $dialog --backtitle "Linux Shell Tutorial"  --title "Calender"  --infobox  "`cat  menuchoice.temp.$$`"  9 25 ; read** |

[http://www.freeos.com/guides/lsst/images/backarr.gif](javascript:history.back(1))

**Answer to if command.**

A) There is file called foo, on your disk and you give command, **$ ./trmfi   foo** what will be output.  
Ans.: foo file will be deleted, and message "foo file deleted" on screen will be printed.

B) If bar file not present on your disk and you give command, $ **./trmfi   bar** what will be output.  
Ans.: Message "rm: cannot remove `bar': No such file or directory" will be printed because bar file does not exist on disk and we have called rm command, so error from rm commad

C) And if you type **$ ./trmfi**, What will be output.  
Ans.:  Following message will be shown by rm command, because rm is called from script without any parameters.  
rm: too few arguments  
Try `rm --help' for more information.

[http://www.freeos.com/guides/lsst/images/backarr.gif](javascript:history.back(1))

**Answer to Variables in Linux.**

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

or

     (b) **$ echo HOME**

Which of the above command is correct & why?

Ans.: (a) command is correct, since we have to print the contains of variable (HOME) and not the HOME. You must use **$**followed by variable name to print variables cotaines.  
  
[http://www.freeos.com/guides/lsst/images/backarr.gif](javascript:history.back(1))

**Answer to Process Section.**

1) Is it example of Multitasking?  
Ans.: Yes, since you are running two process simultaneously.

2) How you will you find out the both running process (MP3 Playing & Letter typing)?  
Ans.: Try **$ ps aux**or**$ ps ax | grep  process-you-want-to-search**

3) "Currently only two Process are running in your Linux/PC environment", Is it True or False?, And how you will verify this?  
Ans.: No its not true, when you start Linux Os, various process start in background for different purpose. To verify this simply use **top** or **ps aux** command.

4) You don't want to listen music (MP3 Files) but want to continue with other work on PC, you will take any of the following action:

1. Turn off Speakers
2. Turn off Computer / Shutdown Linux Os
3. Kill the MP3 playing process
4. None of the above

Ans.: Use action no. 3 i.e. kill the MP3 process.  
**Tip:** First find the PID of MP3 playing process by issuing command:  
**$ ps ax | grep mp3-process-name**  
Then in the first column you will get PID of process. Kill this PID to end the process as:  
**$ kill  PID**

**Or**you can try killall command to kill process by name as follows: **$ killall  mp3-process-name**

[http://www.freeos.com/guides/lsst/images/backarr.gif](javascript:history.back(1))

**Linux Console (Screen)**

How can I write colorful message on Linux Console? , mostly this kind of question is asked by newcomers (Specially those who are learning shell programming!). As you know in Linux everything is considered as a file, our console is one of such special file. You can write special character sequences to console, which control every aspects of the console like Colors on screen, Bold or Blinking text effects, clearing the screen, showing text boxes etc. For this purpose we have to use special code called escape sequence code.  Our Linux console is based on the DEC VT100 serial terminals which support ANSI escape sequence code.

What is special character sequence and how to write it to Console?

By default what ever you send to console it is printed as its. For e.g. consider following echo statement,  
**$ echo "Hello World"**  
Hello World  
Above **echo**statement prints sequence of character on screen, but if there is any special escape sequence (control character) in sequence , then first some action is taken according to escape sequence (or control character) and then normal character is printed on console. For e.g. following echo command prints message in Blue color on console  
**$ echo -e "\033[34m   Hello Colorful  World!"  
Hello Colorful  World!**Above echo statement uses ANSI escape sequence (**\033[34m**), above entire string ( i.e.  **"\033[34m   Hello Colorful  World!"**) is process as follows

1) First **\033**, is escape character, which causes to take some action  
2) Here it set screen foreground color to Blue using **[34m** escape code.  
3) Then it prints our normal message **Hello Colorful  World!** in blue color.

Note that ANSI escape sequence begins with \033 (Octal value) which is represented as **^[**in termcap and terminfo files of terminals and documentation.

You can use **echo** statement to print message, to use ANSI escape sequence you must use **-e** option (switch) with echo statement, general syntax is as follows  
Syntax  
echo   -e  "**\033[***escape-code*    *your-message*"

In above syntax you have to use**\033[**as its with different *escape-code*for different operations. As soon as console receives the message it start to process/read it, and if it found escape character (**\033**) it moves to escape mode, then it read "**[**" character and moves into **Command Sequence Introduction** (CSI) mode. In CSI mode console reads a series of ASCII-coded decimal numbers (know as parameter) which are separated by semicolon (**;**) . This numbers are read until console action letter or character is not found (which determines what action to take). In above example

|  |  |
| --- | --- |
| **\033** | **Escape character** |
| **[** | Start of CSI |
| **34** | 34 is parameter |
| **m** | m is letter (specifies action) |

Following table show important list of such *escape-code/action letter or character*

|  |  |  |
| --- | --- | --- |
| **Character or letter** | **Use in CSI** | **Examples** |
| h | Set the ANSI mode | **echo -e "\033[h"** |
| l | Clears the ANSI mode | **echo -e "\033[l"** |
| m | Useful to show characters in different **colors** or effects such as **BOLD** and Blink, see below for parameter taken by m. | **echo -e  "\033[35m Hello World"** |
| q | Turns keyboard num lock, caps lock, scroll lock LED on or off, see below. | **echo -e "\033[2q"** |
| s | Stores the current cursor x,y position (col , row position) and attributes | **echo -e "\033[7s"** |
| u | Restores cursor position and attributes | **echo -e "\033[8u"** |

m understand following parameters

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Meaning** | **Example** |
| 0 | Sets default color scheme (White foreground and Black background), normal intensity, no blinking etc. |  |
| 1 | Set **BOLD** intensity | **$ echo -e "I am \033[1m BOLD \033[0m Person"** I am **BOLD**Person Prints **BOLD** word in bold intensity and next ANSI Sequence remove bold effect (**\033[0m**) |
| 2 | Set dim intensity | **$ echo -e "\033[1m  BOLD \033[2m DIM  \033[0m"** |
| 5 | Blink Effect | **$ echo -e "\033[5m Flash!  \033[0m"** |
| 7 | Reverse video effect i.e. Black foreground and white background in default color scheme | **$ echo -e "\033[7m Linux OS! Best OS!! \033[0m"** |
| 11 | Shows special control character as graphics character. For e.g. Before issuing this command press alt key (hold down it) from numeric key pad press 178 and leave both key; nothing will be printed. Now give --> command shown in example and try the above, it works. (Hey you must know extended ASCII Character for this!!!) | **$ press alt + 178 $ echo -e "\033[11m" $ press alt + 178 $ echo -e "\033[0m" $ press alt + 178** |
| 25 | Removes/disables blink effect |  |
| 27 | Removes/disables reverse effect |  |
| 30 - 37 | Set foreground color 31 - RED 32 - Green xx - Try to find yourself this left as exercise for you **:-)** | **$ echo -e "\033[31m I am in Red"** |
| 40 - 47 | Set background color xx - Try to find yourself this left as exercise for you **:-)** | **$ echo -e "\033[44m Wow!!!"** |

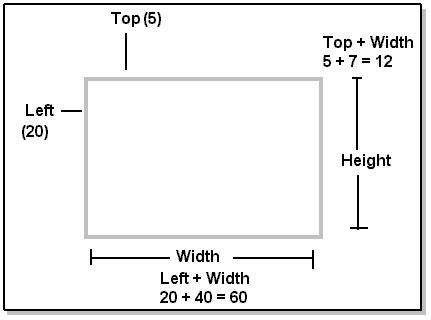
q understand following parameters

|  |  |
| --- | --- |
| **Parameters** | **Meaning** |
| 0 | Turns off all LEDs on Keyboard |
| 1 | Scroll lock LED on and others off |
| 2 | Num lock LED on and others off |
| 3 | Caps lock LED on and others off |

[Click here to see example of q command.](http://www.freeos.com/guides/lsst/datafiles/kbdfuns)

[Click here to see example of m command.](http://www.freeos.com/guides/lsst/datafiles/demom)  
  
[Click here to see example of s and u command.](http://www.freeos.com/guides/lsst/scripts/q17)

This is just quick introduction about Linux Console and what you can do using this Escape sequence. Above table does not contains entire CSI sequences. My up-coming tutorial series on C Programming Language will defiantly have entire story with S-Lang and curses (?). What ever knowledge you gain here will defiantly first step towards the serious programming using c. This much knowledge is sufficient for  Shell Programming, now try the following exercise **:-) I am Hungry give me More Programming Exercise & challenges! :-)**1) Write function box(),  that will draw box on screen (In shell Script)  
    box (left, top, height, width)  
    For e.g. box (20,5,7,40)



Hint: Use ANSI Escape sequence  
1) Use of 11 parameter to m  
2) Use following for cursor movement  
   *row;col***H**  
      or  
   *rowl;col***f**  
    
  For e.g.  
**$ echo   -e "\033[5;10H Hello"  
  $ echo   -e "\033[6;10f Hi"**

In Above example prints Hello message at row 5 and column 6 and Hi at 6th row and 10th Column.

[http://www.freeos.com/guides/lsst/images/backarr.gif](javascript:history.back(1))

**Shell Built in Variables**

|  |  |
| --- | --- |
| **Shell Built in Variables** | **Meaning** |
| **$#** | Number of command line arguments. Useful to test no. of command line args in shell script. |
| **$\*** | All arguments to shell |
| **$@** | Same as above |
| **$-** | Option supplied to shell |
| **$$** | PID of shell |
| **$!** | PID of last started background process (started with &) |

[See example of **$@** and **$\***variable.](http://www.freeos.com/guides/lsst/ch02sec14.html#exaofbuildinsv)

After clicking the above link

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"**

|  |
| --- |
|  |