



Written by Vivek G. Gite.

Cyberciti Computers & nixCraft, Pune, INDIA.

Linux Shell Scripting Tutorial v1.05r3 A Beginner's handbook

Copyright © 1999-2002 by Vivek G. Gite
<vivek@nixcraft.com>



(Formally know as www.vivek-tech.com)

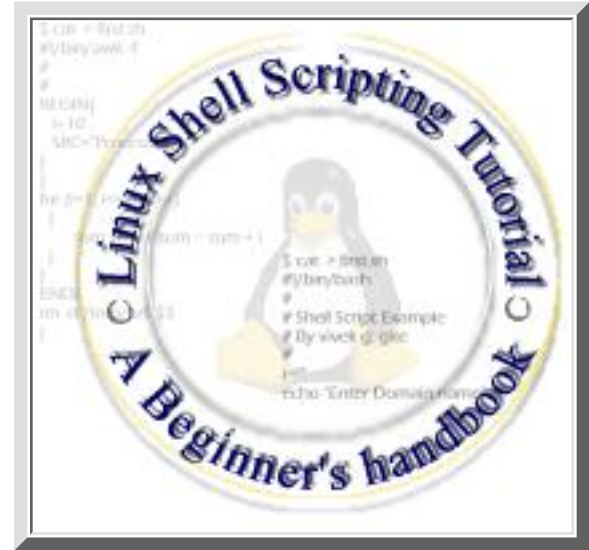


Table of Contents

Chapter 1: [Quick Introduction to Linux](#)

- [What Linux is?](#)
- [Who developed the Linux?](#)
- [How to get Linux?](#)
- [How to Install Linux](#)
- [Where I can use Linux?](#)
- [What Kernel Is?](#)
- [What is Linux Shell?](#)
- [How to use Shell](#)
- [What is Shell Script ?](#)
- [Why to Write Shell Script ?](#)
- [More on Shell...](#)

Chapter 2: [Getting started with Shell Programming](#)

- [How to write shell script](#)
- [Variables in shell](#)
- [How to define User defined variables \(UDV\)](#)
- [Rules for Naming variable name \(Both UDV and System Variable\)](#)
- [How to print or access value of UDV \(User defined variables\)](#)
- [echo Command](#)

[Shell Arithmetic](#)

[More about Quotes](#)

[Exit Status](#)

[The read Statement](#)

[Wild cards \(Filename Shorthand or meta Characters\)](#)

[More commands on one command line](#)

[Command Line Processing](#)

[Why Command Line arguments required](#)

[Redirection of Standard output/input i.e. Input - Output redirection](#)

[Pipes](#)

[Filter](#)

[What is Processes](#)

[Why Process required](#)

[Linux Command\(s\) Related with Process](#)

Chapter 3: Shells (bash) structured Language Constructs

[Decision making in shell script \(i.e. if command\)](#)

[test command or \[expr \]](#)

[if...else...fi](#)

[Nested ifs](#)

[Multilevel if-then-else](#)

[Loops in Shell Scripts](#)

[for loop](#)

[Nested for loop](#)

[while loop](#)

[The case Statement](#)

[How to de-bug the shell script?](#)

Chapter 4: Advanced Shell Scripting Commands

[/dev/null - to send unwanted output of program](#)

[Local and Global Shell variable \(export command\)](#)

[Conditional execution i.e. && and ||](#)

[I/O Redirection and file descriptors](#)

[Functions](#)

[User Interface and dialog utility-Part I](#)

[User Interface and dialog utility-Part II](#)

[Message Box \(msgbox\) using dialog utility](#)

[Confirmation Box \(yesno box\) using dialog utility](#)

[Input \(inputbox\) using dialog utility](#)

[User Interface using dialog Utility - Putting it all together](#)

[trap command](#)

[The shift Command](#)

[getopts command](#)

Chapter 5: [Essential Utilities for Power User](#)

[Preparing for Quick Tour of essential utilities](#)

[Selecting portion of a file using cut utility](#)

[Putting lines together using paste utility](#)

[The join utility](#)

[Translating range of characters using tr utility](#)

[Data manipulation using awk utility](#)

[sed utility - Editing file without using editor](#)

[Removing duplicate lines from text database file using uniq utility](#)

[Finding matching pattern using grep utility](#)

Chapter 6: [Learning expressions with ex](#)

[Getting started with ex](#)

[Printing text on-screen](#)

[Deleting lines](#)

[Copying lines](#)

[Searching the words](#)

[Find and Replace \(Substituting regular expression\)](#)

[Replacing word with confirmation from user](#)

[Finding words](#)

[Using range of characters in regular expressions](#)

[Using & as Special replacement character](#)

[Converting lowercase character to uppercase](#)

Chapter 7: [awk Revisited](#)

[Getting Starting with awk](#)

[Predefined variables of awk](#)

[Doing arithmetic with awk](#)

[User Defined variables in awk](#)

[Use of printf statement](#)

[Use of Format Specification Code](#)

[if condition in awk](#)

[Loops in awk](#)

[Real life examples in awk](#)

[awk miscellaneous](#)

[sed - Quick Introduction](#)

[Redirecting the output of sed command](#)

[How to write sed scripts?](#)

[More examples of sed](#)

Chapter 8: [Examples of Shell Scripts](#)

Logic Development:

[Shell script to print given numbers sum of all digit](#)

[Shell script to print contains of file from given line number to next given number of lines](#)

[Shell script to say Good morning/Afternoon/Evening as you log in to system](#)

[Shell script to find whether entered year is Leap or not](#)

[Sort the given five number in ascending order](#) (use of array)

Command line (args) handling:

[Adding 2 nos. supplied as command line args](#)

[Calculating average of given numbers on command line args](#)

[Finding out biggest number from given three nos supplied as command line args](#)

[Shell script to implement getopt statement.](#)

[Basic math Calculator](#) (case statement)

Loops using while & for loop:

[Print nos. as 5,4,3,2,1 using while loop](#)

[Printing the patterns using for loop.](#)

Arithmetic in shell scripting:

[Performing real number calculation in shell script](#)

[Converting decimal number to hexadecimal number](#)

[Calculating factorial of given number](#)

File handling:

[Shell script to determine whether given file exist or not.](#)

Screen handling/echo command with escape sequence code:

[Shell script to print "Hello World" message, in Bold, Blink effect, and in different colors like red, brown etc.](#)

Background process implementation:

[Digital clock using shell script](#)

User interface and Functions in shell script:

[Shell script to implements menu based system.](#)

System Administration:

[Getting more information about your working environment through shell script](#)

[Shell script to gathered useful system information such as CPU, disks, Ram and your environment etc.](#)

[Shell script to add DNS Entry to BIND Database with default Nameservers, Mail Servers \(MX\) and host](#)

Integrating awk script with shell script:

[Script to convert file names from UPPERCASE to lowercase file names or vice versa.](#)

Chapter 9: [Other Resources](#)

[Appendix - A](#) : Linux File Server Tutorial (LFST) version b0.1 Rev. 2

[Appendix - B](#) : Linux Command Reference (LCR)

[About the author](#)

[About this Document](#)

[Home](#)

[Next](#)

[Up](#)

Quick Introduction to Linux

(Cyberciti Computers & nixCraft has years of experince in Linux / Unix / FreeBSD. If you need any assistance, education, support for Linux / Unix, write to sales@cyberciti.biz)

Introduction

This tutorial is designed for beginners who wish to learn the basics of shell scripting/programming plus introduction to power tools such as awk, sed, etc. It is not help or manual for the shell; while reading this tutorial you can find manual quite useful (type `man bash` at `$` prompt to see manual pages). Manual contains all necessary information you need, but it won't have that much examples, which makes idea more clear. For this reason, this tutorial contains examples rather than all the features of shell.

Audience for this tutorial

I assumes you have at least working knowledge of Linux i.e. basic commands like how to create, copy, remove files/directories etc or how to use editor like vi or mcedit and login to your system. But not expects any programming language experience. If you have access to Linux, this tutorial will provide you an easy-to-follow introduction to shell scripting.

What's different about this tutorial

Many other tutorial and books on Linux shell scripting are either too basic, or skips important intermediate steps. But this tutorial, maintained the balance between these two. It covers the many real life modern example of shell scripting which are almost missed by many other tutorials/documents/books. I have used a hands-on approach in this tutorial. The idea is very clear "*do it yourself or learn by doing*" i.e. trying things yourself is the best way to learn, so examples are presented as complete working shell scripts, which can be typed in and executed

Chapter Organization

Chapter 1 to 4 shows most of the useful and important shell scripting concepts. Chapter 5 introduction to tools & utilities which can be used while programming the Linux shell smartly. Chapter 6 and 7 is all about expression and expression mostly used by tools such as sed and awk. Chapter 8 is loaded with tons of shell scripting examples divided into different categories. Chapter 9 gives more resources information which can be used while learning the shell scripting like information on Linux file system, common Linux command reference and other resources.

Chapter 1 introduces to basic concepts such as what is Linux, where Linux can used and continue explaining the shell, shell script and kernel etc.

Chapter 2 shows how to write the shell script and execute them. It explains many basic concepts which requires to write shell script.

Chapter 3 is all about making decision in shell scripting as well as loops in shell. It explains what expression are, how shell understands the condition/decisions. It also shows you nesting concept for if and for loop statement and debugging of shell script.

Chapter 4 introduces the many advanced shell scripting concepts such as function, user interface, File Descriptors, signal handling, Multiple command line arguments etc.

Chapter 5 introduces to powerful utility programs which can be used variety of purpose while programming the shell.

Chapter 6 and 7 gives more information on patterns, filters, expressions, and off course sed and awk is covered in depth.

Chapter 8 contains lot of example of shell scripting divided into various category such as logic development, system administration etc.

Note that  indicates advanced shell scripting concepts, you can skip this if you are really new to Linux or Programming, though this is not RECOMMENDED by me.

I hope you get as much pleasure reading this tutorial, as I had writing it. After reading this tutorial if you are able to write your own powerful shell scripts, then I think the purpose of writing this tutorial is served and finally if you do get time after reading this tutorial drop me an e-mail message about your comment/suggestion/questions and off course bugs (errors) you find regarding this tutorial.

[Prev](#)

[Home](#)
[Up](#)

[Next](#)
What Linux is?

What Linux is?







- [Free](#)
 - [Unix Like](#)
 - [Open Source](#)
 - Network operating system
-

Who developed the Linux?

In 1991, Linus Torvalds studying Unix at the University, where he used special educational experimental purpose operating system called Minix (small version of Unix and used in Academic environment). But Minix had it's own limitations. Linus felt he could do better than the Minix. So he developed his own version of Minix, which is now know as Linux. Linux is Open Source From the start of the day. For more information on Linus Torvalds, please visit [his home page](#).

How to get Linux?

Linux available for download over the net, this is useful if your internet connection is fast. Another way is order the CD-ROMs which saves time, and the installation from CD-ROM is fast/automatic. Various Linux distributions available. Following are important Linux distributions.

Linux distributions.	Website/Logo
Red Hat Linux: http://www.redhat.com/	
SuSE Linux: http://www.suse.com/	
Mandrake Linux: http://www.mandrakesoft.com/	
Caldera Linux: http://www.calderasystems.com/	
Debian GNU/Linux: http://www.debian.org/	
Slackware Linux: http://www.slackware.com/	

Note: If you are in India then you can get Linux Distribution from the Leading Computer magazine such as [PC Quest](#) (Even PCQuest has got its own Linux flavour) or if you are in Pune, India please visit [our web site](#) to obtained the Red Hat Linux or any other official Linux distribution. Note that you can also obtained your Linux distribution with Linux books which you purchase from local book store.

[Prev](#)

Who developed the Linux?

[Home](#)

[Up](#)

[Next](#)

How to Install Linux

How to Install Linux ?

Please visit the [LESSBS](#) Project home page for Quick Visual Installation Guide for Red Hat Linux version 6.2 and 7.2.

Where I can use Linux?

You can use Linux as Server Os or as stand alone Os on your PC. (But it is best suited for Server.) As a server Os it provides different services/network resources to client. Server Os must be:

- Stable
- Robust
- Secure
- High Performance

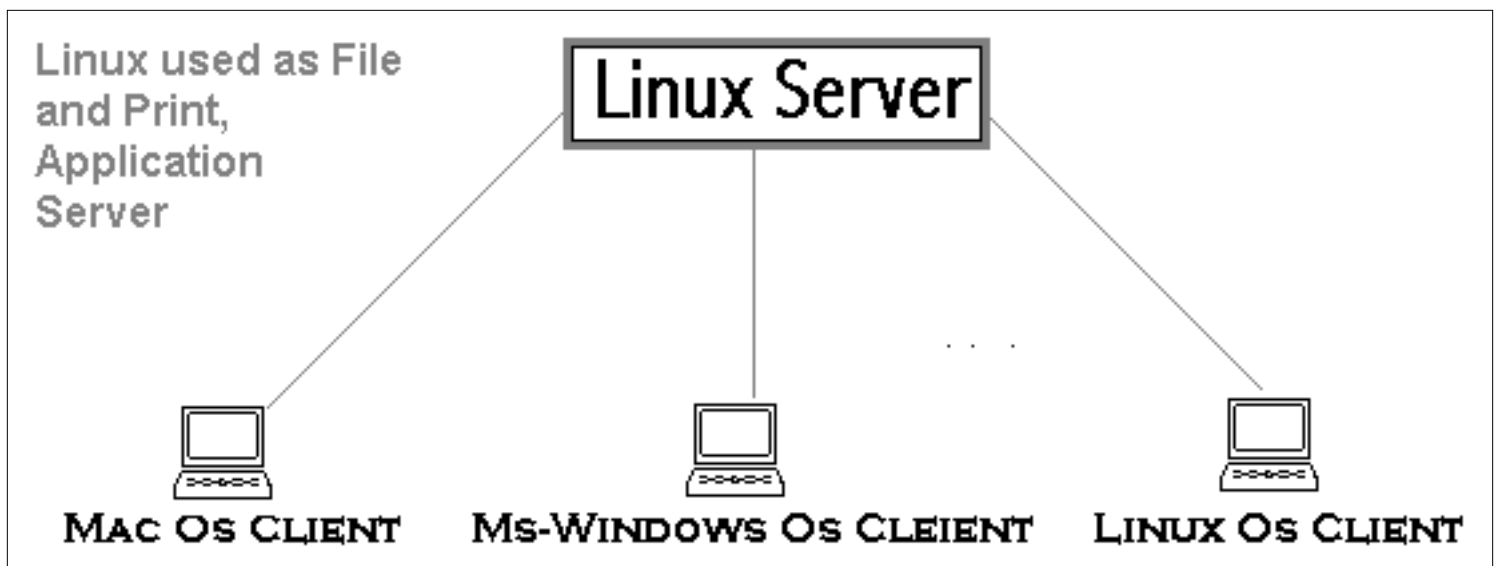
Linux offers all of the above characteristics plus its Open Source and Free OS. So Linux can be used as:

(1) On *stand alone workstation/PC* for word processing, graphics, software development, internet, e-mail, chatting, small personal database management system etc.

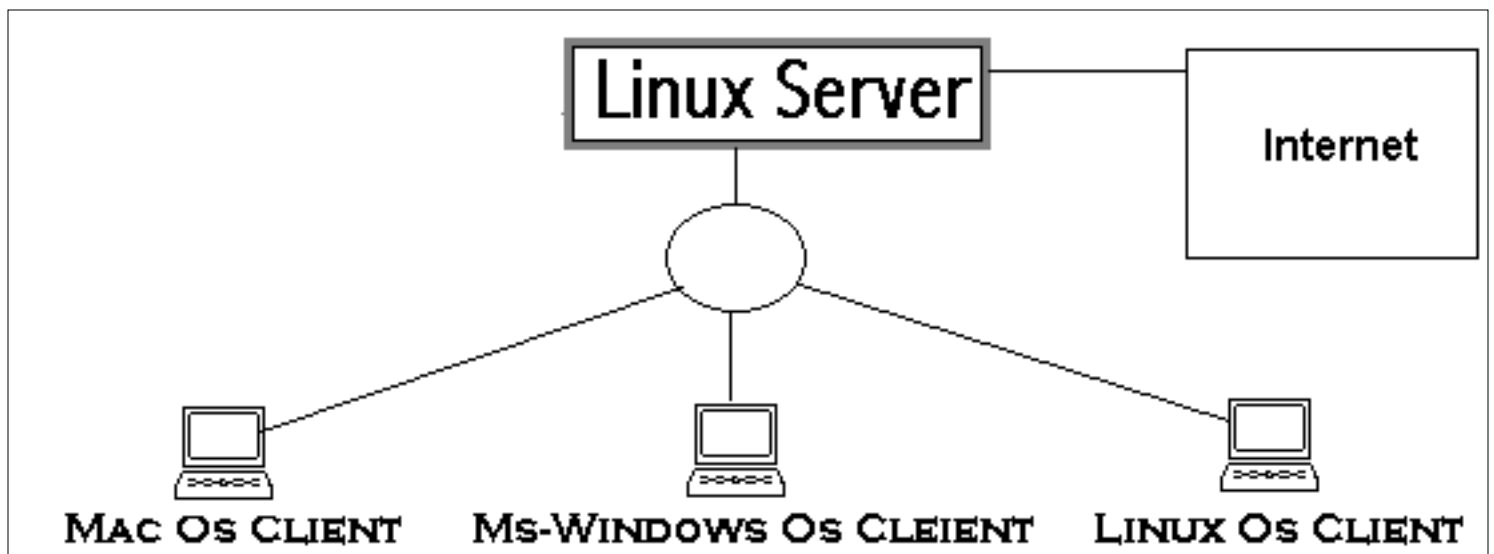
(2) In *network environment* as:

(A) *File and Print or Application Server*

Share the data, Connect the expensive device like printer and share it, e-mail within the LAN/intranet etc are some of the application.



(B) Linux sever can be connected to Internet, So that PC's on intranet can share the internet/e-mail etc. You can put your web sever that run your web site or transmit the information on the internet.



Linux Server can act as Proxy/Mail/WWW/Router Server etc.

So you can use Linux for:

- Personal Work
- Web Server
- Software Development Workstation
- Workgroup Server
- In Data Center for various server activities such as FTP, Telnet, SSH, Web, Mail, Proxy, Proxy Cache Appliance etc

See the [LESSBS](#) project for more information on Linux Essential Services (as mentioned above) and how to implement them in easy manner for you or your organization.

[Prev](#)

How to Install Linux

[Home](#)

[Up](#)

[Next](#)

What Kernel Is?

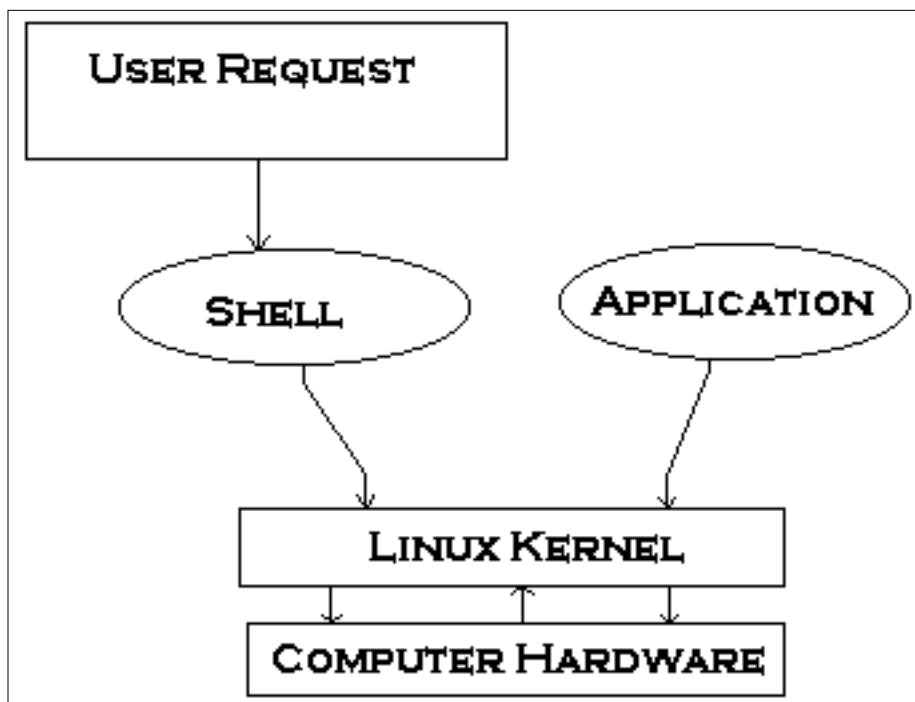
What Kernel Is?

Kernel is heart of Linux Os.

It manages resource of Linux Os. Resources means facilities available in Linux. For e.g. Facility to store data, print data on printer, memory, file management etc .

Kernel decides who will use this resource, for how long and when. It runs your programs (or set up to execute binary files).

The kernel acts as an intermediary between the computer hardware and various programs/application/shell.



It's Memory resident portion of Linux. It performance following task :-

- I/O management
- Process management
- Device management
- File management
- Memory management

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 passed to kernel.

Shell is a user program or it's a 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

[Prev](#)

What Kernel Is?

[Home](#)

[Up](#)

[Next](#)

How to use Shell

How to use Shell

To use shell (You start to use your shell as soon as you log into your system) you have to simply type commands.

See common [Linux Command](#) for syntax and example, this can be used as quick reference while programming the 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.
-

Which Shell We are using to write Shell Script ?

In this tutorial we are using bash shell.

Objective of this Tutorial (LSST v.1.5)

Try to understand Linux Os

Try to understand the basics of Linux shell

Try to learn the Linux shell programming

What I need to learn this Tutorial (LSST v.1.5)

Linux OS (I have used Red Hat Linux distribution Version 6.x+)

Web Browse to read tutorial. (IE or Netscape) For PDF version you need PDF reader.

Linux - bash shell. (Available with almost all Linux Distributions. By default bash is default shell for Red Hat Linux Distribution). All the scripts are also tested on Red Hat Linux version 7.2.

Getting started with Shell Programming

In this part of tutorial you are introduced to shell programming, how to write script, execute them etc. We will be 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](#) for more information.
- Basic command navigation

Domain Name Rs.445/- p.a.

[Get Cyberciti Domain for Just Rs.445 with 2 Free e-mail]

Linux Shell Scripting Tutorial (LSST) v1.05r3
Chapter 2: Getting started with Shell Programming

[Prev](#)

[Next](#)

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](#) , 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. <i>Syntax:</i> # 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 <i>syntax:</i> echo "Message"

 [How Shell Locates the file](#) (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 its 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](#) for more information.

[Prev](#)

Getting started with Shell Programming

[Home](#)

[Up](#)

[Next](#)

Variables in Shell

[Advertisement]

Domain Name Rs.445/- p.a.

[[Get Cyberciti Domain for Just Rs.445 with 2 Free e-mail](#)]

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.](#)

Caution: Do not modify System variable this can some time create problems.

[Prev](#)

How to write shell script

[Home](#)

[Up](#)

[Next](#)

How to define User defined variables
(UDV)

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 define User defined variables
(UDV)

[Up](#)

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

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](#)

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

[Up](#)

echo Command

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"**

 How to display colorful text on screen with bold or blink effects, how to print text on any row, column on screen, [click here for more!](#)

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
```

 See [Parameter substitution - To save your time.](#)

[Prev](#)

echo Command

[Home](#)

[Up](#)

[Next](#)

More about Quotes

More about Quotes

There are three types of quotes

Quotes	Name	Meaning
"	Double Quotes	"Double Quotes" - Anything enclosed 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 $?
```

💡 **\$?** useful variable, want to know more such Linux variables [click here](#) to explore them!

[Prev](#)

More about Quotes

[Home](#)

[Up](#)

[Next](#)

The read Statement

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.	<code>\$ ls *</code>	will show all files
		<code>\$ ls a*</code>	will show all files whose first name is starting with letter 'a'
		<code>\$ ls *.c</code>	will show all files having extension .c
		<code>\$ ls ut*.c</code>	will show all files having extension .c but file name must begin with 'ut'.
?	Matches any single character.	<code>\$ ls ?</code>	will show all files whose names are 1 character long
		<code>\$ ls fo?</code>	will show all files whose names are 3 character long and file name begin with fo
[...]	Matches any one of the enclosed characters	<code>\$ ls [abc]*</code>	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
```

[Prev](#)

The read Statement

[Home](#)

[Up](#)

[Next](#)

More command on one command line

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.

[Prev](#)

More commands on one command line

[Home](#)

[Up](#)

[Next](#)

Why Command Line arguments required

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 tell 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}

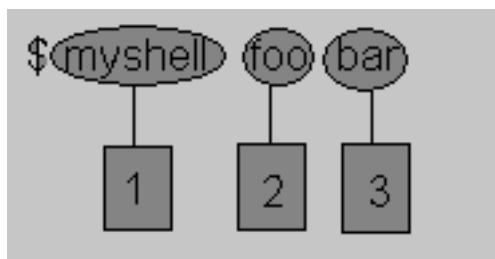
Here rm is command and filename is file which you would like to remove. This way you tell rm command which file you would like to remove. So we are doing one way communication with our command by specifying filename. Also you can pass command line arguments to your script to make it more users friendly. 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



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

2 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 $#"
```

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 assign 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"
```

[Prev](#)

[Home](#)

[Next](#)

Command Line Processing

[Up](#)

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

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

Mostly all commands give 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

 [Click here to learn more about I/O Redirection](#)

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 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
```

[Prev](#)

Why Command Line arguments required

[Home](#)

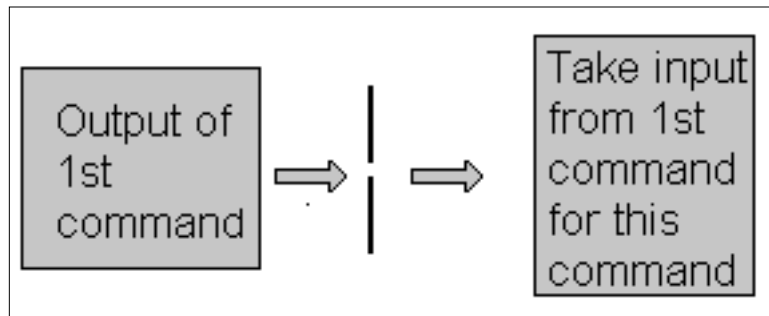
[Up](#)

[Next](#)

Pipe

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`

Examples:

Command using Pipes	Meaning or Use of Pipes
<code>\$ ls more</code>	Output of ls command is given as input to more command So that output is printed one screen full page at a time.
<code>\$ who sort</code>	Output of who command is given as input to sort command So that it will print sorted list of users
<code>\$ who sort > user_list</code>	Same as above except output of sort is send to (redirected) user_list file
<code>\$ who wc -l</code>	Output of who command is given as input to wc command So that it will print number of user who logon to system
<code>\$ ls -l wc -l</code>	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)

[Prev](#)

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

[Home](#)

[Up](#)

[Next](#)

Filter

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](#) is filter which takes its input from sort command and passes this lines as input to uniq; Then unqiqs output is redirected to "u_sname" file.

What is Processes

Process is kind of program or task carried out by your PC. For e.g.

\$ ls -lR

ls command or a request to list files in a directory and all subdirectory in your current directory - It is a process.

Process defined as:

"A process is program (command given by user) to perform specific Job. In Linux when you start process, it gives a number to process (called PID or process-id), PID starts from 0 to 65535."

Why Process required

As You know Linux is multi-user, multitasking Os. It means you can run more than two process simultaneously if you wish. For e.g. To find how many files do you have on your system you may give command like:

```
$ ls / -R | wc -l
```

This command will take lot of time to search all files on your system. So you can run such command in Background or simultaneously by giving command like

```
$ ls / -R | wc -l &
```

The **ampersand (&)** at the end of command tells shells start process (**ls / -R | wc -l**) and run it in background takes next command immediately.

Process & PID defined as:

*"An instance of running command is called **process** and the number printed by shell is called **process-id (PID)**, this PID can be use to refer specific running process."*

Linux Command Related with Process

Following tables most commonly used command(s) with process:

For this purpose	Use this Command	Examples*
To see currently running process	ps	\$ ps
To stop any process by PID i.e. to kill process	kill {PID}	\$ kill 1012
To stop processes by name i.e. to kill process	killall {Process-name}	\$ killall httpd
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 &
To display the owner of the processes along with the processes	ps aux	\$ ps aux
To see if a particular process is running or not. For this purpose you have to use ps command in combination with the grep command	ps ax grep process-U-want-to see	For e.g. you want to see whether Apache web server process is running or not then give command \$ ps ax grep httpd
To see currently running processes and other information like memory and CPU usage with real time updates.	top See the output of top command.	\$ top Note that to exit from top command press q.
To display a tree of processes	pstree	\$ pstree

* To run some of this command you need to be root or equivalent user.

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.

Exercise:

You are working on your Linux workstation (might be learning LSST or some other work like sending mails, typing letter), while doing this work you have started to play MP3 files on your workstation. Regarding this situation, answer the following question:

- 1) Is it example of Multitasking?
- 2) How you will you find out the both running process (MP3 Playing & Letter typing)?
- 3) "Currently only two Process are running in your Linux/PC environment", Is it True or False?, And how you will verify this?
- 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

[Click here for answers.](#)

[Prev](#)

[Home](#)

[Next](#)

Why Process required

[Up](#)

Shells (bash) structured Language
Constructs

Introduction

Making decision is important part in ONCE life as well as in computers logical driven program. In fact logic is not LOGIC until you use decision making. This chapter introduces to the bash's structured language constructs such as:

- Decision making
- Loops

Is there any difference making decision in Real life and with Computers? Well real life decision are quite complicated to all of us and computers even don't have that much power to understand our real life decisions. What computer know is 0 (zero) and 1 that is Yes or No. To make this idea clear, lets play some game (WOW!) with bc - Linux calculator program.

\$ bc

After this command bc is started and waiting for your 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

See what happened if you type $5 > 2$ as follows

5 > 2

1

1 (One?) is response of bc, How? bc compare 5 with 2 as, Is 5 is greater then 2, (If I ask same question to you, your answer will be YES), bc gives this 'YES' answer by showing 1 value. Now try

5 < 2

0

0 (Zero) indicates the false i.e. Is 5 is less than 2?, Your answer will be no which is indicated by bc by showing 0 (Zero). Remember in bc, [relational expression](#) always returns **true** (1) or **false** (0 - zero).

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
$5 > 12$	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

It means when ever there is any type of comparison in Linux Shell It gives only two answer one is YES and NO is other.

In Linux Shell Value	Meaning	Example
Zero Value (0)	Yes/True	0
NON-ZERO Value	No/False	-1, 32, 55 anything but not zero

Remember both bc and Linux Shell uses *different ways to show True/False values*

Value	Shown in bc as	Shown in Linux Shell as
True/Yes	1	0
False/No	0	Non - zero value

[Prev](#)
Linux Command(s) Related with Process

[Home](#)
[Up](#)

[Next](#)
if condition

if condition

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

Syntax:

```

if condition
then
    command1 if condition is true or if exit status
    of condition is 0 (zero)
    ...
    ...
fi
  
```

Condition is defined as:

"Condition is nothing but comparison between two values."

For compression you can use test or [expr] statements or even exist status can be also used.

Expression is defined as:

"An expression is nothing but combination of values, relational operator (such as >, <, <> etc) and mathematical operators (such as +, -, / etc)."

Following are all examples of expression:

```

5 > 2
3 + 6
3 * 65
a < b
c > 5
c > 5 + 30 -1
  
```

Type following commands (assumes you have file called **foo**)

\$ cat foo

\$ echo \$?

The cat command return zero(0) i.e. exit status, 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
```

Run above script as:

```
$ chmod 755 showfile
```

```
$/showfile foo
```

Shell script name is showfile (\$0) and foo is argument (which is \$1). Then shell compare it as follows: if cat \$1 which is expanded to if cat foo.

Detailed explanation

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.

Exercise

Write shell script as follows:

```
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 755 trmif
```

Answer the following question in reference to above script:

- (A) foo file exists 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?

[For Answer click here.](#)

Shells (bash) structured Language Constructs

[Up](#)

test command or [expr]

test command or [expr]

test command or [expr] is used to see if an expression is true, and if it is true it return zero(0), otherwise returns nonzero for false.

Syntax:

test expression OR [expression]

Example:

Following script determine whether given argument number is positive.

```
$ cat > ispositive
#!/bin/sh
#
# Script to see whether argument is positive
#
if test $1 -gt 0
then
echo "$1 number is positive"
fi
```

Run it as follows

\$ chmod 755 ispositive

\$ ispositive 5

5 number is positive

\$ ispositive -45

Nothing is printed

\$ ispositive

./ispositive: test: -gt: unary operator expected

Detailed explanation

The line, if test \$1 -gt 0 , test to see if first command line argument(\$1) is greater than 0. If it is true(0) then test will return 0 and output will printed as 5 number is positive but for -45 argument there is no output because our condition is not true(0) (no -45 is not greater than 0) hence echo statement is skipped. And for last statement we have not supplied any argument hence error ./ispositive: test: -gt: unary operator expected, is generated by shell , to avoid such error we can test whether command line argument is supplied or not.

test or [expr] works with

- 1.Integer (Number without decimal point)
- 2.File types
- 3.Character strings

For Mathematics, use following operator in Shell Script

Mathematical Operator in Shell Script	Meaning	Normal Arithmetical/ Mathematical Statements	But in Shell	
			For test statement with if command	For [expr] statement with if command
-eq	is equal to	5 == 6	if test 5 -eq 6	if [5 -eq 6]
-ne	is not equal to	5 != 6	if test 5 -ne 6	if [5 -ne 6]
-lt	is less than	5 < 6	if test 5 -lt 6	if [5 -lt 6]
-le	is less than or equal to	5 <= 6	if test 5 -le 6	if [5 -le 6]
-gt	is greater than	5 > 6	if test 5 -gt 6	if [5 -gt 6]
-ge	is greater than or equal to	5 >= 6	if test 5 -ge 6	if [5 -ge 6]

NOTE: == is equal, != is not equal.

For string Comparisons use

Operator	Meaning
string1 = string2	string1 is equal to string2
string1 != string2	string1 is NOT equal to string2
string1	string1 is NOT NULL or not defined
-n string1	string1 is NOT NULL and does exist
-z string1	string1 is NULL and does exist

Shell also test for file and directory types

Test	Meaning
-s file	Non empty file
-f file	Is File exist or normal file and not a directory
-d dir	Is Directory exist and not a file
-w file	Is writeable file
-r file	Is read-only file
-x file	Is file is executable

Logical Operators

Logical operators are used to combine two or more condition at a time

Operator	Meaning
! expression	Logical NOT
expression1 -a expression2	Logical AND

expression1 -o expression2

Logical OR

[Prev](#)

Decision making in shell script (i.e. if command)

[Home](#)

[Up](#)

[Next](#)

if...else...fi

if...else...fi

If given condition is true then command1 is executed otherwise command2 is executed.

Syntax:

```
if condition
then
    condition is zero (true - 0)
    execute all commands up to else statement

else
    if condition is not true then
    execute all commands up to fi

fi
```

For e.g. Write Script as follows:

```
$ vi isnump_n
#!/bin/sh
#
# Script to see whether argument is positive or negative
#
if [ $# -eq 0 ]
then
echo "$0 : You must give/supply one integers"
exit 1
fi

if test $1 -gt 0
then
echo "$1 number is positive"
else
echo "$1 number is negative"
fi
```

Try it as follows:

```
$ chmod 755 isnump_n
```

```
$ isnump_n 5
```

```
5 number is positive
```

```
$ isnump_n -45
```


-45 number is negative

\$ isnump_n

./ispos_n : You must give/supply one integers

\$ isnump_n 0

0 number is negative

Detailed explanation

First script checks whether command line argument is given or not, if not given then it print error message as *./ispos_n : You must give/supply one integers*. if statement checks whether number of argument (\$#) passed to script is not equal (-eq) to 0, if we passed any argument to script then this if statement is false and if no command line argument is given then this if statement is true. The echo command i.e.

```
echo "$0 : You must give/supply one integers"
```

```

|           |
|           |
1           2

```

1 will print Name of script

2 will print this error message

And finally statement exit 1 causes normal program termination with exit status 1 (nonzero means script is not successfully run).

The last sample run **\$ isnump_n 0** , gives output as *"0 number is negative"*, because given argument is not > 0, hence condition is false and it's taken as negative number. To avoid this replace second if statement with **if test \$1 -ge 0**.

💡 Nested if-else-fi

You can write the entire if-else construct within either the body of the if statement or the body of an else statement. This is called the nesting of ifs.

```

$ vi nestedif.sh
osch=0

echo "1. Unix (Sun Os)"
echo "2. Linux (Red Hat)"
echo -n "Select your os choice [1 or 2]? "
read osch

if [ $osch -eq 1 ] ; then

    echo "You Pick up Unix (Sun Os)"

else #### nested if i.e. if within if #####

```

```

if [ $osch -eq 2 ] ; then
    echo "You Pick up Linux (Red Hat)"
else
    echo "What you don't like Unix/Linux OS."
fi

```

```
fi
```

Run the above shell script as follows:

```
$ chmod +x nestedif.sh
```

```
$ ./nestedif.sh
```

```
1. Unix (Sun Os)
```

```
2. Linux (Red Hat)
```

```
Select you os choice [1 or 2]? 1
```

```
You Pick up Unix (Sun Os)
```

```
$ ./nestedif.sh
```

```
1. Unix (Sun Os)
```

```
2. Linux (Red Hat)
```

```
Select you os choice [1 or 2]? 2
```

```
You Pick up Linux (Red Hat)
```

```
$ ./nestedif.sh
```

```
1. Unix (Sun Os)
```

```
2. Linux (Red Hat)
```

```
Select you os choice [1 or 2]? 3
```

```
What you don't like Unix/Linux OS.
```

Note that Second *if-else* construct is nested in the first *else* statement. If the condition in the first *if* statement is false the the condition in the second *if* statement is checked. If it is false as well the final *else* statement is executed.

You can use the nested *ifs* as follows also:

Syntax:

```

if condition
then
    if condition
    then
        .....
        ..
        do this
    else
        ....
        ..
        do this
    fi
else

```

```
        ...  
        .....  
        do this  
fi
```

[Prev](#)

test command or [expr]

[Home](#)

[Up](#)

[Next](#)

Multilevel if-then-else

Multilevel if-then-else

Syntax:

```

if condition
then
    condition is zero (true - 0)
    execute all commands up to elif statement
elif condition1
then
    condition1 is zero (true - 0)
    execute all commands up to elif statement
elif condition2
then
    condition2 is zero (true - 0)
    execute all commands up to elif statement
else
    None of the above condtion,condtion1,condtion2 are true (i.e.
    all of the above nonzero or false)
    execute all commands up to fi
fi

```

For multilevel if-then-else statement try the following script:

```

$ cat > elf
#
#!/bin/sh
# Script to test if..elif...else
#
if [ $1 -gt 0 ]; then
    echo "$1 is positive"
elif [ $1 -lt 0 ]
then
    echo "$1 is negative"
elif [ $1 -eq 0 ]
then
    echo "$1 is zero"
else
    echo "Opps! $1 is not number, give number"
fi

```

Try above script as follows:

```
$ chmod 755 elf
```

```
$ ./elf 1
```

```
$ ./elf -2
```

```
$ ./elf 0
```

```
$ ./elf a
```

Here o/p for last sample run:

```
./elf: [: -gt: unary operator expected
```

```
./elf: [: -lt: unary operator expected
```

```
./elf: [: -eq: unary operator expected
```

```
Opps! a is not number, give number
```

Above program gives error for last run, here integer comparison is expected therefore error like `./elf: [: -gt: unary operator expected` occurs, but still our program notify this error to user by providing message `Opps! a is not number, give number`.

[Prev](#)

if...else...fi

[Home](#)

[Up](#)

[Next](#)

Loops in Shell Scripts

Loops in Shell Scripts

Loop defined as:

"Computer can repeat particular instruction again and again, until particular condition satisfies. A group of instruction that is executed repeatedly is called a loop."

Bash supports:

- for loop
- while loop

Note that in each and every loop,

- (a) First, the variable used in loop condition must be initialized, then execution of the loop begins.
- (b) A test (condition) is made at the beginning of each iteration.
- (c) The body of loop ends with a statement that modifies the value of the test (condition) variable.

for Loop

Syntax:

```

for { variable name } in { list }
do
    execute one for each item in the list until the list is
    not finished (And repeat all statement between do and done)
done

```

Before try to understand above syntax try the following script:

```

$ cat > testfor
for i in 1 2 3 4 5
do
echo "Welcome $i times"
done

```

Run it above script as follows:

```

$ chmod +x testfor
$ ./testfor

```

The for loop first creates i variable and assigned a number to i from the list of number from 1 to 5, The shell execute echo statement for each assignment of i. (This is usually know as iteration) This process will continue until all the items in the list were not finished, because of this it will repeat 5 echo statements. To make you idea more clear try following script:

```

$ cat > mtable
#!/bin/sh
#
#Script to test for loop
#
#
if [ $# -eq 0 ]
then
echo "Error - Number missing form command line argument"
echo "Syntax : $0 number"
echo "Use to print multiplication table for given number"
exit 1
fi
n=$1
for i in 1 2 3 4 5 6 7 8 9 10
do
echo "$n * $i = `expr $i \* $n`"
done

```

Save above script and run it as:

```

$ chmod 755 mtable
$ ./mtable 7
$ ./mtable

```

For first run, above script print multiplication table of given number where i = 1,2 ... 10 is multiply by given n (here

command line argument 7) in order to produce multiplication table as

```
7 * 1 = 7
7 * 2 = 14
...
..
7 * 10 = 70
```

And for second test run, it will print message -

Error - Number missing form command line argument

Syntax : ./mtable number

Use to print multiplication table for given number

This happened because we have not supplied given number for which we want multiplication table, Hence script is showing Error message, Syntax and usage of our script. This is good idea if our program takes some argument, let the user know what is use of the script and how to used the script.

Note that to terminate our script we used 'exit 1' command which takes 1 as argument (1 indicates error and therefore script is terminated)

Even you can use following syntax:

Syntax:

```
for (( expr1; expr2; expr3 ))
do
    .....
    ...
    repeat all statements between do and
    done until expr2 is TRUE
Done
```

In above syntax BEFORE the first iteration, **expr1** is evaluated. This is usually used to initialize variables for the loop. All the statements between do and done is executed repeatedly UNTIL the value of **expr2** is TRUE. AFTER each iteration of the loop, **expr3** is evaluated. This is usually use to increment a loop counter.

```
$ cat > for2
for (( i = 0 ; i <= 5; i++ ))
do
    echo "Welcome $i times"
done
```

Run the above script as follows:

```
$ chmod +x for2
```

```
$ ./for2
```

```
Welcome 0 times
Welcome 1 times
Welcome 2 times
Welcome 3 times
Welcome 4 times
Welcome 5 times
```

In above example, first expression ($i = 0$), is used to set the value variable **i** to zero.

Second expression is condition i.e. all statements between do and done executed as long as expression 2 (i.e continue as long as the value of variable **i** is less than or equal to 5) is TRUE.

Last expression **i++** increments the value of **i** by 1 i.e. it's equivalent to $i = i + 1$ statement.

💡 Nesting of for Loop

As you see the [if statement can nested](#), similarly loop statement can be nested. You can nest the for loop. To understand the nesting of for loop see the following shell script.

```
$ vi nestedfor.sh
for (( i = 1; i <= 5; i++ ))      ### Outer for loop ###
do

    for (( j = 1 ; j <= 5; j++ )) ### Inner for loop ###
    do
        echo -n "$i "
    done

    echo "" ##### print the new line ###
done
```

Run the above script as follows:

```
$ chmod +x nestedfor.sh
```

```
$ ./nestedfor.sh
```

```
1 1 1 1 1
2 2 2 2 2
3 3 3 3 3
4 4 4 4 4
5 5 5 5 5
```

Here, for each value of **i** the inner loop is cycled through 5 times, with the variable **j** taking values from 1 to 5. The inner for loop terminates when the value of **j** exceeds 5, and the outer loop terminates when the value of **i** exceeds 5.

Following script is quite interesting, it prints the chess board on screen.

```
$ vi chessboard
for (( i = 1; i <= 9; i++ )) ### Outer for loop ###
do
    for (( j = 1 ; j <= 9; j++ )) ### Inner for loop ###
    do
        tot=`expr $i + $j`
        tmp=`expr $tot % 2`
        if [ $tmp -eq 0 ]; then
            echo -e -n "\033[47m "
        else
            echo -e -n "\033[40m "
        fi
    done
    echo -e -n "\033[40m" ##### set back background colour to black
    echo "" ##### print the new line ###
done
```

Run the above script as follows:

```
$ chmod +x chessboard
```

```
$ ./chessboard
```

On my terminal above script produce the output as follows:

```

root@ls:/home/vivek/scripts/new
[root@ls new]# ./chessboard
  [root@ls new]#

```

Above shell script can be explained as follows:

Command(s)/Statements	Explanation
for ((i = 1; i <= 9; i++)) do	Begin the outer loop which runs 9 times., and the outer loop terminates when the value of i exceeds 9
for ((j = 1 ; j <= 9; j++)) do	Begins the inner loop, for each value of i the inner loop is cycled through 9 times, with the variable j taking values from 1 to 9. The inner for loop terminates when the value of j exceeds 9.
tot=`expr \$i + \$j` tmp=`expr \$tot % 2`	See for even and odd number positions using these statements.
if [\$tmp -eq 0]; then echo -e -n "\033[47m " else echo -e -n "\033[40m " fi	If even number position print the white colour block (using echo -e -n "\033[47m " statement); otherwise for odd position print the black colour box (using echo -e -n "\033[40m " statement). These statements are responsible to print entire chess board on screen with alternate colours.
done	End of inner loop
echo -e -n "\033[40m "	Make sure its black background as we always have on our terminals.
echo ""	Print the blank line
done	End of outer loop and shell scripts get terminated by printing the chess board.

Exercise

Try to understand the [shell scripts \(for loops\) shown in exercise chapter](#).

[Prev](#)

Loops in Shell Scripts

[Home](#)

[Up](#)

[Next](#)

while loop

while loop

Syntax:

```
while [ condition ]
do
    command1
    command2
    command3
    ..
    ....
done
```

Loop is executed as long as given condition is true. For e.g.. [Above for loop program](#) (shown in last section of for loop) can be written using while loop as:

```
$cat > nt1
#!/bin/sh
#
#Script to test while statement
#
#
if [ $# -eq 0 ]
then
    echo "Error - Number missing form command line argument"
    echo "Syntax : $0 number"
    echo " Use to print multiplication table for given number"
exit 1
fi
n=$1
i=1
while [ $i -le 10 ]
do
    echo "$n * $i = `expr $i \* $n`"
    i=`expr $i + 1`
done
```

Save it and try as

```
$ chmod 755 nt1
```

```
$/nt1 7
```

Above loop can be explained as follows:

n=\$1	Set the value of command line argument to variable n. (Here it's set to 7)
i=1	Set variable i to 1
while [\$i -le 10]	This is our loop condition, here if value of i is less than 10 then, shell execute all statements between do and done
do	Start loop
echo "\$n * \$i = `expr \$i * \$n`"	Print multiplication table as $7 * 1 = 7$ $7 * 2 = 14$ $7 * 10 = 70$, Here each time value of variable n is multiply be i.
i=`expr \$i + 1`	Increment i by 1 and store result to i. (i.e. $i=i+1$) Caution: If you ignore (remove) this statement than our loop become infinite loop because value of variable i always remain less than 10 and program will only output $7 * 1 = 7$ E (infinite times)
done	Loop stops here if i is not less than 10 i.e. condition of loop is not true. Hence loop is terminated.

[Prev](#)
for loop

[Home](#)
[Up](#)

[Next](#)
The case Statement

The case Statement

The case statement is good alternative to Multilevel if-then-else-fi statement. It enable you to match several values against one variable. Its easier to read and write.

Syntax:

```
case $variable-name in
    pattern1)    command
                ...
                ..
                command;;
    pattern2)    command
                ...
                ..
                command;;
    patternN)    command
                ...
                ..
                command;;
    *)           command
                ...
                ..
                command;;
esac
```

The *\$variable-name* is compared against the patterns until a match is found. The shell then executes all the statements up to the two semicolons that are next to each other. The default is *) and its executed if no match is found. For e.g. write script as follows:

```
$ cat > car
#
# if no vehicle name is given
# i.e. -z $1 is defined and it is NULL
#
# if no command line arg
if [ -z $1 ]
then
    rental="*** Unknown vehicle ***"
elif [ -n $1 ]
then
# otherwise make first arg as rental
```

```

rental=$1
fi

case $rental in
    "car") echo "For $rental Rs.20 per k/m" ;;
    "van") echo "For $rental Rs.10 per k/m" ;;
    "jeep") echo "For $rental Rs.5 per k/m" ;;
    "bicycle") echo "For $rental 20 paisa per k/m" ;;
    *) echo "Sorry, I can not gat a $rental for you" ;;
esac

```

Save it by pressing CTRL+D and run it as follows:

```
$ chmod +x car
```

```
$ car van
```

```
$ car car
```

```
$ car Maruti-800
```

First script will check, that if \$1(first command line argument) is given or not, if NOT given set value of rental variable to "*** Unknown vehicle ***",if command line arg is supplied/given set value of rental variable to given value (command line arg). The \$rental is compared against the patterns until a match is found.

For first test run its match with van and it will show output *"For van Rs.10 per k/m."*

For second test run it print, *"For car Rs.20 per k/m"*.

And for last run, there is no match for Maruti-800, hence default i.e. *) is executed and it prints, *"Sorry, I can not gat a Maruti-800 for you"*.

Note that esac is always required to indicate end of case statement.

See the one more [example of case](#) statement in chapter 4 of section shift command.

[Prev](#)

while loop

[Home](#)

[Up](#)

[Next](#)

How to de-bug the shell script?

How to de-bug the shell script?

While programming shell sometimes you need to find the errors (bugs) in shell script and correct the errors (remove errors - debug). For this purpose you can use `-v` and `-x` option with `sh` or `bash` command to debug the shell script. General syntax is as follows:

Syntax:

```
sh option { shell-script-name }
```

OR

```
bash option { shell-script-name }
```

Option can be

-v Print shell input lines as they are read.

-x After expanding each simple-command, bash displays the expanded value of `PS4` system variable, followed by the command and its expanded arguments.

Example:

```
$ cat > dsh1.sh
#
# Script to show debug of shell
#
tot=`expr $1 + $2`
echo $tot
```

Press `ctrl + d` to save, and run it as

```
$ chmod 755 dsh1.sh
```

```
$ ./dsh1.sh 4 5
```

```
9
```

```
$ sh -x dsh1.sh 4 5
```

```
#
# Script to show debug of shell
```

```
#
```

```
tot=`expr $1 + $2`
```

```
expr $1 + $2
```

```
++ expr 4 + 5
```

```
+ tot=9
```

```
echo $tot
```

```
+ echo 9
```

```
9
```

See the above output, `-x` shows the exact values of variables (or statements are shown on screen with values).

\$ sh -v dsh1.sh 4 5

Use -v option to debug complex shell script.

[Prev](#)

The case Statement

[Home](#)

[Up](#)

[Next](#)

Advanced Shell Scripting

Introduction

After learning basis of shell scripting, its time to learn more advance features of shell scripting/command such as:

- Functions
- User interface
- Conditional execution
- File Descriptors
- traps
- Multiple command line args handling etc

How to de-bug the shell script?

/dev/null - to send unwanted output of program

/dev/null - Use to send unwanted output of program

This is special Linux file which is used to send any unwanted output from program/command.

Syntax:

```
command > /dev/null
```

Example:

```
$ ls > /dev/null
```

Output of above command is not shown on screen its send to this special file. The /dev directory contains other device files. The files in this directory mostly represent peripheral devices such disks like floppy disk, sound card, line printers etc. See the [file system tutorial](#) for more information on Linux disk, partition and file system.

Future Point:

Run the following two commands

```
$ ls > /dev/null
```

```
$ rm > /dev/null
```

1) Why the output of last command is not redirected to /dev/null device?

[Prev](#)

Advanced Shell Scripting Commands

[Home](#)[Up](#)[Next](#)

Local and Global Shell variable (export command)

Local and Global Shell variable (export command)

Normally all our variables are local. Local variable can be used in same shell, if you load another copy of shell (by typing the **/bin/bash** at the \$ prompt) then new shell ignored all old shell's variable. For e.g.

Consider following example

```
$ vech=Bus
$ echo $vech
Bus
$ /bin/bash
$ echo $vech
```

NOTE:-Empty line printed

```
$ vech=Car
$ echo $vech
Car
$ exit
$ echo $vech
Bus
```

Command	Meaning
\$ vech=Bus	Create new local variable 'vech' with Bus as value in first shell
\$ echo \$vech	Print the contains of variable vech
\$ /bin/bash	Now load second shell in memory (Which ignores all old shell's variable)
\$ echo \$vech	Print the contains of variable vech
\$ vech=Car	Create new local variable 'vech' with Car as value in second shell
\$ echo \$vech	Print the contains of variable vech
\$ exit	Exit from second shell return to first shell
\$ echo \$vech	Print the contains of variable vech (Now you can see first shells variable and its value)

Global shell defined as:

"You can copy old shell's variable to new shell (i.e. first shells variable to seconds shell), such variable is know as Global Shell variable."

To set global variable you have to use export command.

Syntax:

```
export variable1, variable2,.....variableN
```

Examples:

\$ vech=Bus

\$ echo \$vech

Bus

\$ export vech

\$ /bin/bash

\$ echo \$vech

Bus

\$ exit

\$ echo \$vech

Bus

Command	Meaning
\$ vech=Bus	Create new local variable 'vech' with Bus as value in first shell
\$ echo \$vech	Print the contains of variable vech
\$ export vech	Export first shells variable to second shell i.e. global variable
\$ /bin/bash	Now load second shell in memory (Old shell's variable is accessed from second shell, <i>if they are exported</i>)
\$ echo \$vech	Print the contains of variable vech
\$ exit	Exit from second shell return to first shell
\$ echo \$vech	Print the contains of variable vech

[Prev](#)

/dev/null - to send unwanted output of program

[Home](#)

[Up](#)

[Next](#)

Conditional execution i.e. && and ||

Conditional execution i.e. && and ||

The control operators are && (read as AND) and || (read as OR). The syntax for AND list is as follows

Syntax:

```
command1 && command2
```

command2 is executed if, and only if, command1 returns an exit status of zero.

The syntax for OR list as follows

Syntax:

```
command1 || command2
```

command2 is executed if and only if command1 returns a non-zero exit status.

You can use both as follows

Syntax:

```
command1 && comamnd2 if exist status is zero || command3 if exit status is non-zero
```

if command1 is executed successfully then shell will run command2 and if command1 is not successful then command3 is executed.

Example:

```
$ rm myf && echo "File is removed successfully" || echo "File is not removed"
```

If file (myf) is removed successful (exist status is zero) then "*echo File is removed successfully*" statement is executed, otherwise "*echo File is not removed*" statement is executed (since exist status is non-zero)

[Prev](#)

Local and Global Shell variable (export command)

[Home](#)[Up](#)[Next](#)

I/O Redirection and file descriptors

I/O Redirection and file descriptors

As you know I/O redirectors are used to send output of command to file or to read input from file.

Consider following example

```
$ cat > myf
```

```
This is my file
```

```
^D (press CTRL + D to save file)
```

Above command send output of cat command to myf file

```
$ cal
```

Above command prints calendar on screen, but if you wish to store this calendar to file then give command

```
$ cal > mycal
```

The cal command send output to mycal file. This is called *output redirection*.

```
$ sort
```

```
10
```

```
-20
```

```
11
```

```
2
```

```
^D
```

```
-20
```

```
2
```

```
10
```

```
11
```

sort command takes input from keyboard and then sorts the number and prints (send) output to screen itself. If you wish to take input from file (for sort command) give command as follows:

```
$ cat > nos
```

```
10
```

```
-20
```

```
11
```

```
2
```

```
^D
```

```
$ sort < nos
```

```
-20
```

```
2
```

```
10
```

```
11
```

First you created the file *nos* using cat command, then *nos* file given as input to *sort* command which prints sorted numbers. This is called *input redirection*.

In Linux (And in C programming Language) your keyboard, screen etc are all treated as files. Following are name of such files

Standard File	File Descriptors number	Use	Example
stdin	0	as Standard input	Keyboard
stdout	1	as Standard output	Screen
stderr	2	as Standard error	Screen

By default in Linux every program has three files associated with it, (when we start our program these three files are automatically opened by your shell). The use of first two files (i.e. stdin and stdout) , are already seen by us. The last file stderr (numbered as 2) is used by our program to print error on screen. You can redirect the output from a file descriptor directly to file with following syntax

Syntax:

file-descriptor-number>filename

Examples: (Assume the file **bad_file_name111** does not exists)

\$ rm bad_file_name111

rm: cannot remove `bad_file_name111': No such file or directory

Above command gives error as output, since you don't have file. Now if we try to redirect this error-output to file, it can not be send (redirect) to file, try as follows:

\$ rm bad_file_name111 > er

Still it prints output on stderr as *rm: cannot remove `bad_file_name111': No such file or directory*, And if you see er file as **\$ cat er** , this file is empty, since output is send to error device and you can not redirect it to copy this error-output to your file 'er'. To overcome this problem you have to use following command:

\$ rm bad_file_name111 2>er

Note that no space are allowed between 2 and >, The 2>er directs the standard error output to file. 2 number is default number (file descriptors number) of stderr file. To clear your idea consider another example by writing shell script as follows:

```
$ cat > demoscrcr
if [ $# -ne 2 ]
then
    echo "Error : Number are not supplied"
    echo "Usage : $0 number1 number2"
    exit 1
fi
ans=`expr $1 + $2`
echo "Sum is $ans"
```

Run it as follows:

\$ chmod 755 demoscrcr

\$./demoscrr

Error : Number are not supplied

Usage : ./demoscrr number1 number2


```
$ ./demoscr > er1
```

```
$ ./demoscr 5 7
```

```
Sum is 12
```

For first sample run , our script prints error message indicating that you have not given two number.

For second sample run, you have redirected output of script to file er1, since it's error we have to show it to user, It means we have to print our error message on stderr not on stdout. To overcome this problem replace above echo statements as follows

```
echo "Error : Number are not supplied" 1>&2
```

```
echo "Usage : $0 number1 number2" 1>&2
```

Now if you run it as follows:

```
$ ./demoscr > er1
```

```
Error : Number are not supplied
```

```
Usage : ./demoscr number1 number2
```

It will print error message on stderr and not on stdout. The **1>&2** at the end of echo statement, directs the standard output (stdout) to standard error (stderr) device.

Syntax:

from>&destination

[Prev](#)

Conditional execution i.e. && and ||

[Home](#)

[Up](#)

[Next](#)

Functions

Functions

Humans are intelligent animals. They work together to perform all of life's task, in fact most of us depend upon each other. For e.g. you rely on milkman to supply milk, or teacher to learn new technology (if computer teacher). What all this mean is you can't perform all of life's task alone. You need somebody to help you to solve specific task/problem.

The above logic also applies to computer program (shell script). When program gets complex we need to use divide and conquer technique. It means whenever programs gets complicated, we divide it into small chunks/entities which are known as *functions*.

Function is series of instruction/commands. Function performs particular activity in shell i.e. it had specific work to do or simply say task. To define function use following syntax:

Syntax:

```
function-name ( )
{
    command1
    command2
    . . . . .
    . . .
    commandN
    return
}
```

Where function-name is name of you function, that executes series of commands. A return statement will terminate the function. *Example:*

Type SayHello() at \$ prompt as follows

```
$ SayHello()
{
  echo "Hello $LOGNAME, Have nice computing"
  return
}
```

To execute this SayHello() function just type it name as follows:

```
$ SayHello
```

```
Hello vivek, Have nice computing.
```

This way you can call function. Note that after restarting your computer you will loss this SayHello() function, since its created for current session only. To overcome this problem and to add you own function to automate some of the day today life task, add your function to */etc/bashrc* file. To add function to this file you must logon as root. Following is the sample */etc/bashrc* file with *today()* function , which is used to print formatted date. First logon as root or if you already logon with your name (your login is not root), and want to move to root account, then you can type following command , when asked

for password type root (administrators) password

\$ su -l

password:

Open file **/etc/bashrc** using vi and goto the end of file (by pressing shift+G) and type the today() function:

```
# vi /etc/bashrc
# At the end of file add following in /etc/bashrc file
#
# today() to print formatted date
#
# To run this function type today at the $ prompt
# Added by Vivek to show function in Linux
#
today()
{
echo This is a `date +"%A %d in %B of %Y (%r)"`
return
}
```

Save the file and exit it, after all this modification your file may look like as follows (type command **cat /etc/bashrc**)

```
# cat /etc/bashrc
# /etc/bashrc

# System wide functions and aliases
# Environment stuff goes in /etc/profile

# For some unknown reason bash refuses to inherit
# PS1 in some circumstances that I can't figure out.
# Putting PS1 here ensures that it gets loaded every time.

PS1="[u@\h \W]\\$ "

#
# today() to print formatted date
#
# To run this function type today at the $ prompt
# Added by Vivek to show function in Linux
today()
{
echo This is a `date +"%A %d in %B of %Y (%r)"`
return
}
```

To run function first completely logout by typing exit at the \$ prompt (Or press CTRL + D, Note you may have to type exit (CTRL +D) twice if you login to root account by using su command) ,then login and type \$ today , this way today() is available to all user in your system, If you want to add particular

function to particular user then open `.bashrc` file in users home directory as follows:

```
# vi .bashrc
OR
# mcedit .bashrc
At the end of file add following in .bashrc file
SayBuy()
{
echo "Buy $LOGNAME ! Life never be the same, until you login again!"
echo "Press a key to logout. . ."
read
return
}
```

Save the file and exit it, after all this modification your file may look like as follows (type command **cat .bashrc**)

```
# cat .bashrc
# .bashrc
#
# User specific aliases and functions
# Source global definitions

if [ -f /etc/bashrc ]; then
. /etc/bashrc
fi

SayBuy()
{
echo "Buy $LOGNAME ! Life never be the same, until you login again!"
echo "Press a key to logout. . ."
read
return
}
```

To run function first logout by typing exit at the \$ prompt (Or press CTRL + D) ,then logon and type \$ SayBuy , this way SayBuy() is available to only in your login and not to all user in system, Use `.bashrc` file in your home directory to add User specific aliases and functions only.

Tip: If you want to show some message or want to perform some action when you logout, Open file `.bash_logout` in your home directory and add your stuff here For e.g. When ever I logout, I want to show message Buy! Then open your `.bash_logout` file using text editor such as vi and add statement:

```
echo "Buy $LOGNAME, Press a key. . ."
read
```

Save and exit from the file. Then to test this logout from your system by pressing CTRL + D (or type exit) immediately you will see message "Buy xxxxx, Press a key. . .", after pressing key you will be logout and login prompt will be shown to you. :-)

Why to write function?

- Saves lot of time.
- Avoids rewriting of same code again and again
- Program is easier to write.
- Program maintains is very easy.

 [Passing parameters to User define function.](#)

[Prev](#)

I/O Redirection and file descriptors

[Home](#)

[Up](#)

[Next](#)

User Interface and dialog utility

User Interface and dialog utility-Part I

Good program/shell script must interact with users. You can accomplish this as follows:

- (1) Use command line arguments (args) to script when you want interaction i.e. pass command line args to script as : **\$./sutil.sh foo 4**, where *foo* & *4* are command line args passed to shell script *sutil.sh*.
- (2) Use statement like *echo* and *read* to read input into variable from the prompt. For e.g. Write script as:

```
$ cat > userinte
#
# Script to demo echo and read command for user interaction
#
echo "Your good name please : "
read na
echo "Your age please : "
read age
neyr=`expr $age + 1`
echo "Hello $na, next year you will be $neyr yrs old."
```

Save it and run as

```
$ chmod 755 userinte
```

```
$ ./userinte
```

```
Your good name please :
```

```
Vivek
```

```
Your age please :
```

```
25
```

```
Hello Vivek, next year you will be 26 yrs old.
```

Even you can create menus to interact with user, first show menu option, then ask user to choose menu item, and take appropriate action according to selected menu item, this technique is show in following script:

```

$ cat > menuui
#
# Script to create simple menus and take action according to that
selected
# menu item
#
while :
do
clear
echo "-----"
echo " Main Menu "
echo "-----"
echo "[1] Show Todays date/time"
echo "[2] Show files in current directory"
echo "[3] Show calendar"
echo "[4] Start editor to write letters"
echo "[5] Exit/Stop"
echo "======"
echo -n "Enter your menu choice [1-5]: "
read yourch
case $yourch in
1) echo "Today is `date` , press a key. . ." ; read ;;
2) echo "Files in `pwd`" ; ls -l ; echo "Press a key. . ." ;
read ;;
3) cal ; echo "Press a key. . ." ; read ;;
4) vi ;;
5) exit 0 ;;
*) echo "Oops!!! Please select choice 1,2,3,4, or 5";
echo "Press a key. . ." ; read ;;
esac
done

```

Above all statement explained in following table:

Statement	Explanation
while :	Start infinite loop, this loop will only break if you select 5 (i.e. Exit/Stop menu item) as your menu choice
do	Start loop
clear	Clear the screen, each and every time

<pre>echo "-----" echo " Main Menu " echo "-----" echo "[1] Show Todays date/time" echo "[2] Show files in current directory" echo "[3] Show calendar" echo "[4] Start editor to write letters" echo "[5] Exit/Stop" echo "=====</pre>	Show menu on screen with menu items
<pre>echo -n "Enter your menu choice [1-5]: "</pre>	Ask user to enter menu item number
<pre>read yourch</pre>	Read menu item number from user
<pre>case \$yourch in 1) echo "Today is `date` , press a key. ..." ; read ;; 2) echo "Files in `pwd`" ; ls -l ; echo "Press a key. ..." ; read ;; 3) cal ; echo "Press a key. ..." ; read ;; 4) vi ;; 5) exit 0 ;; *) echo "Opps!!! Please select choice 1,2,3,4, or 5"; echo "Press a key. ..." ; read ;; esac</pre>	Take appropriate action according to selected menu item, If menu item is not between 1 - 5, then show error and ask user to input number between 1-5 again
<pre>done</pre>	Stop loop , if menu item number is 5 (i.e. Exit/Stop)

User interface usually includes, menus, different type of boxes like info box, message box, Input box etc. In Linux shell (i.e. bash) there is no built-in facility available to create such user interface, But there is one utility supplied with Red Hat Linux version 6.0 called dialog, which is used to create different type of boxes like info box, message box, menu box, Input box etc. Next section shows you how to use dialog utility.

[Prev](#)
Functions

[Home](#)
[Up](#)

[Next](#)
User Interface and dialog utility-Part II

User Interface and dialog utility-Part II

Before programming using dialog utility you need to install the dialog utility, since dialog utility is not installed by default.

For Red Hat Linux 6.2 user install the dialog utility as follows (First insert Red Hat Linux 6.2 CD into CDROM drive)

```
# mount /mnt/cdrom
# cd /mnt/cdrom/RedHat/RPMS
# rpm -ivh dialog-0.6-16.i386.rpm
```

For Red Hat Linux 7.2 user install the dialog utility as follows (First insert Red Hat Linux 7.2 # 1 CD into CDROM drive)

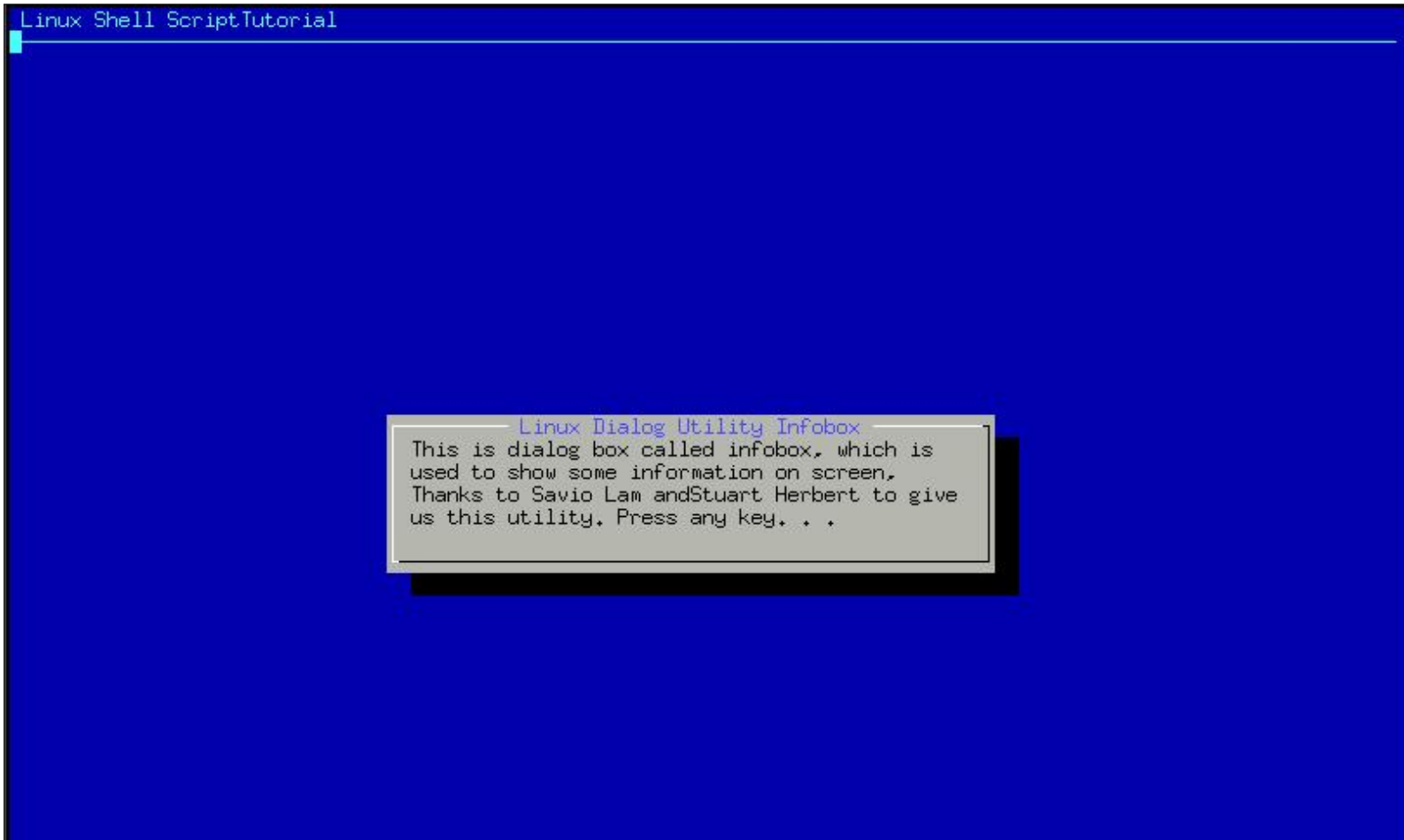
```
# mount /mnt/cdrom
# cd /mnt/cdrom/RedHat/RPMS
# rpm -ivh dialog-0.9a-5.i386.rpm
```

After installation you can start to use dialog utility. Before understanding the syntax of dialog utility try the following script:

```
$ cat > dial
dialog --title "Linux Dialog Utility Infobox" --backtitle "Linux Shell
Script\
Tutorial" --infobox "This is dialog box called infobox, which is used
\
to show some information on screen, Thanks to Savio Lam and\
Stuart Herbert to give us this utility. Press any key. . ." 7 50 ;
read
```


Save the shell script and run it as:

```
$ chmod +x dial
$ ./dial
```



```
Linux Shell ScriptTutorial

Linux Dialog Utility Infobox
This is dialog box called infobox, which is
used to show some information on screen.
Thanks to Savio Lam and Stuart Herbert to
give us this utility. Press any key. . .
```



After executing this dialog statement you will see box on screen with titled as "Welcome to Linux Dialog Utility" and message "This is dialog....Press any key. . ." inside this box. The title of box is specified by --title option and infobox with --infobox "Message" with this option. Here 7 and 50 are height-of-box and width-of-box respectively. "Linux Shell Script Tutorial" is the backtitle of dialog show on upper left side of screen and below that line is drawn. Use dialog utility to Display dialog boxes from shell scripts.

Syntax:

```
dialog --title {title} --backtitle {backtitle} {Box options}
where Box options can be any one of following
--yesno      {text}  {height} {width}
--msgbox     {text}  {height} {width}
--infobox    {text}  {height} {width}
--inputbox   {text}  {height} {width} [{init}]
--textbox    {file}  {height} {width}
--menu       {text}  {height} {width} {menu} {height} {tag1} item1}...
```

[Prev](#)

[Home](#)

[Next](#)

User Interface and dialog utility-Part I

[Up](#)

Message Box (msgbox) using dialog utility

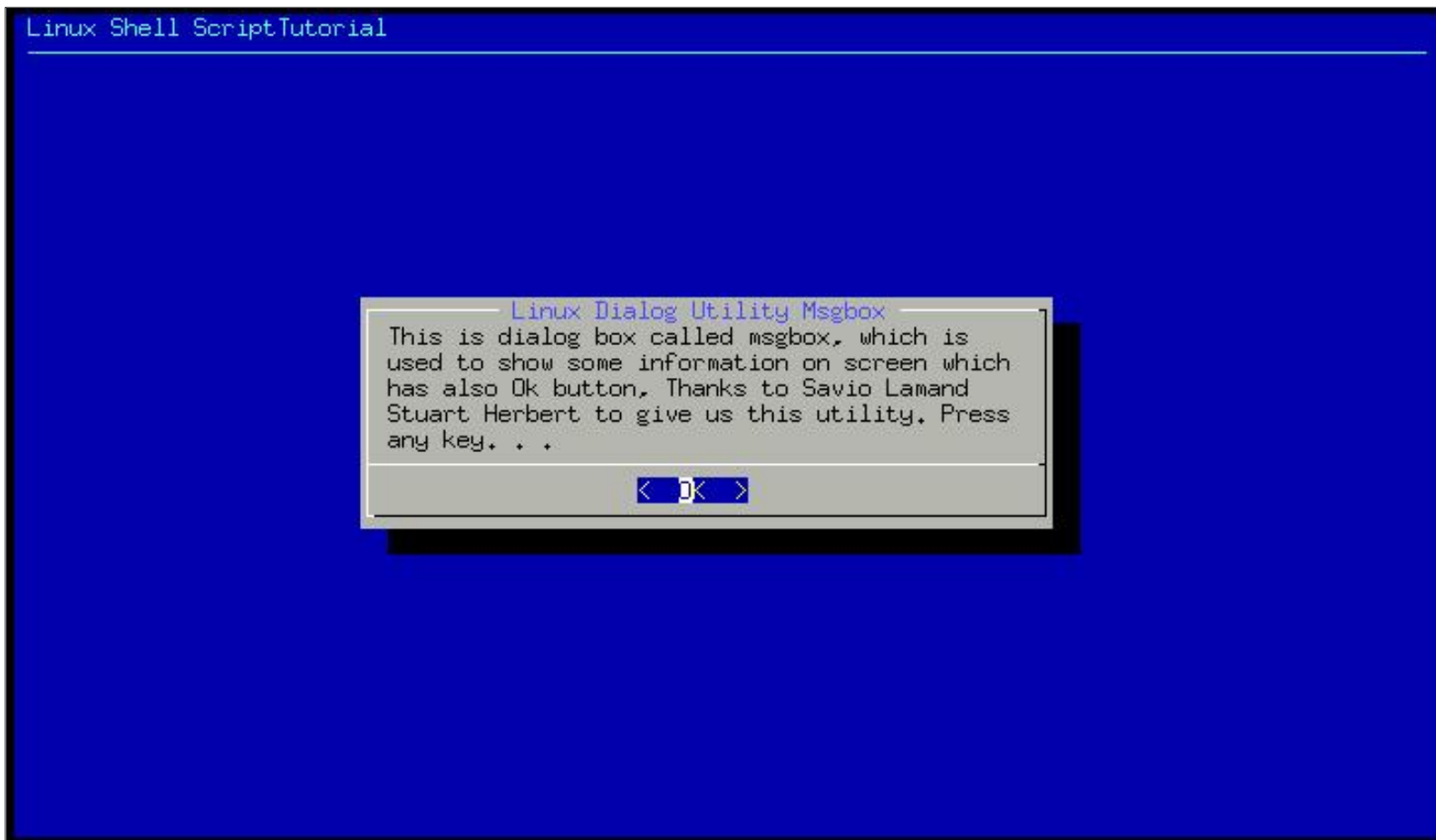
Message box (msgbox) using dialog utility

```
$cat > dia2
dialog --title "Linux Dialog Utility Msgbox" --backtitle "Linux Shell
Script\
Tutorial" --msgbox "This is dialog box called msgbox, which is used\
to show some information on screen which has also Ok button, Thanks to
Savio Lam\
and Stuart Herbert to give us this utility. Press any key. . . " 9 50
```

Save it and run as

```
$ chmod +x dia2
```

```
$ ./dia2
```



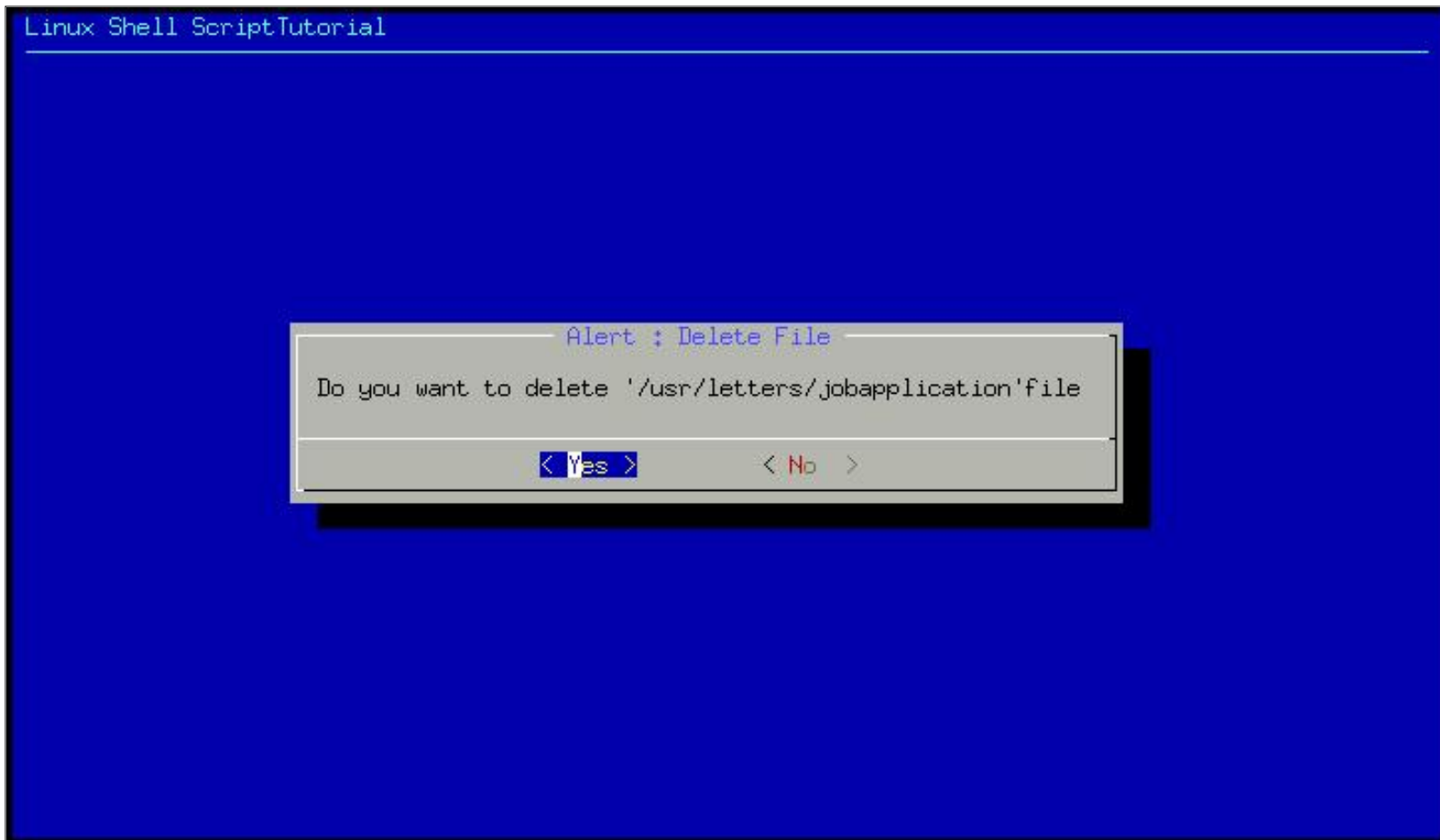
yesno box using dialog utility

```
$ cat > dia3
dialog --title "Alert : Delete File" --backtitle "Linux Shell Script\
Tutorial" --yesno "\nDo you want to delete '/usr/letters/jobapplication'\
file" 7 60
sel=$?
case $sel in
  0) echo "User select to delete file";;
  1) echo "User select not to delete file";;
  255) echo "Canceled by user by pressing [ESC] key";;
esac
```

Save the script and run it as:

```
$ chmod +x dia3
```

```
$ ./dia3
```



Above script creates yesno type dialog box, which is used to ask some questions to the user , and answer to those question either yes or no. After asking question how do we know, whether user has press yes or no button ? The answer is exit status, if user press yes button exit status will be zero, if user press no button exit status will be one and if user press Escape key to cancel dialog box exit status will be one 255. That is what we have tested in our above shell script as

Statement	Meaning
sel=\$?	Get exit status of dialog utility

```
case $sel in
```

```
 0) echo "You select to delete file";;
```

```
 1) echo "You select not to delete file";;
```

```
255) echo "Canceled by you by pressing [Escape] key";;
```

```
esac
```

Now take action according to exit status of dialog utility, if exit status is 0 , delete file, if exit status is 1 do not delete file and if exit status is 255, means Escape key is pressed.

[Prev](#)

Message Box (msgbox) using dialog utility

[Home](#)

[Up](#)

[Next](#)

Input (inputbox) using dialog utility

Input Box (inputbox) using dialog utility

```
$ cat > dia4
dialog --title "Inputbox - To take input from you" --backtitle "Linux
Shell\
Script Tutorial" --inputbox "Enter your name please" 8 60
2>/tmp/input.$$

sel=$?

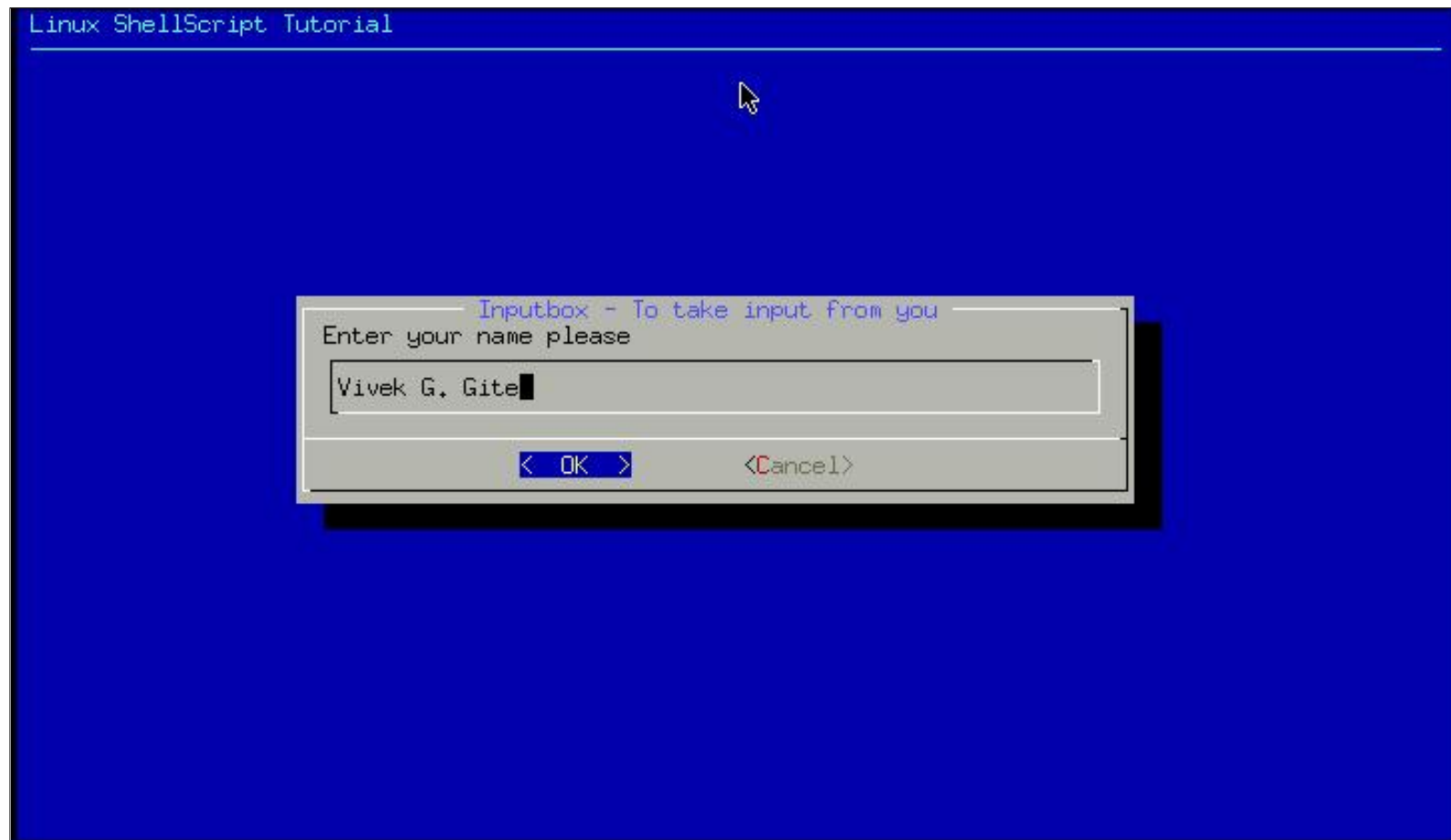
na=`cat /tmp/input.$$`
case $sel in
  0) echo "Hello $na" ;;
  1) echo "Cancel is Press" ;;
  255) echo "[ESCAPE] key pressed" ;;
esac

rm -f /tmp/input.$$
```

Run it as follows:

```
$ chmod +x dia4
```

```
$ ./dia4
```



Inputbox is used to take input from user, In this example we are taking Name of user as input. But where we are going to store

inputted name, the answer is to redirect inputted name to file via statement `2>/tmp/input.$$` at the end of dialog command, which means send screen output to file called `/tmp/input.$$`, later we can retrieve this inputted name and store to variable as follows `na=`cat /tmp/input.$$``.

For input box's exit status refer the following table:

Exit Status for Input box	Meaning
0	Command is successful
1	Cancel button is pressed by user
255	Escape key is pressed by user

[Prev](#)

Confirmation Box (yesno box) using dialog utility

[Home](#)

[Up](#)

[Next](#)

User Interface using dialog Utility - Putting it all together

User Interface using dialog Utility - Putting it all together

Its time to write script to create menus using dialog utility, following are menu items

Date/time

Calendar

Editor

and action for each menu-item is follows :

MENU-ITEM	ACTION
Date/time	Show current date/time
Calendar	Show calendar
Editor	Start vi Editor

```
$ cat > smenu
#
#How to create small menu using dialog
#
dialog --backtitle "Linux Shell Script Tutorial " --title "Main\
Menu" --menu "Move using [UP] [DOWN],[Enter] to\
Select" 15 50 3\
Date/time "Shows Date and Time"\
Calendar "To see calendar "\
Editor "To start vi editor " 2>/tmp/menuitem.$$

menuitem=`cat /tmp/menuitem.$$`

opt=$?

case $menuitem in
Date/time) date;;
Calendar) cal;;
Editor) vi;;
esac
```

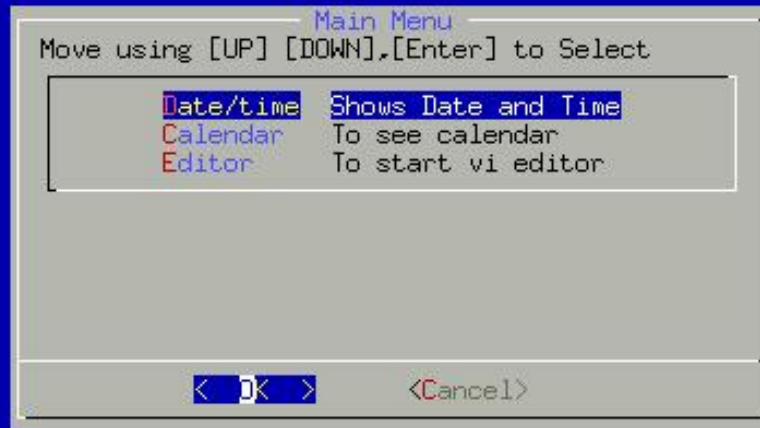
Save it and run as:

```
$ rm -f /tmp/menuitem.$$
```

```
$ chmod +x smenu
```

```
$ ./smenu
```


Linux Shell Script Tutorial



--menu option is used of dialog utility to create menus, menu option take

--menu options	Meaning
"Move using [UP] [DOWN],[Enter] to Select"	This is text shown before menu
15	Height of box
50	Width of box
3	Height of menu
Date/time "Shows Date and Time"	First menu item called as <i>tag1</i> (i.e. Date/time) and description for menu item called as <i>item1</i> (i.e. " Shows Date and Time ")
Calendar "To see calendar "	First menu item called as <i>tag2</i> (i.e. Calendar) and description for menu item called as <i>item2</i> (i.e. " To see calendar ")
Editor "To start vi editor "	First menu item called as <i>tag3</i> (i.e. Editor) and description for menu item called as <i>item3</i> (i.e. " To start vi editor ")
2>/tmp/menuitem.\$\$	Send selected menu item (tag) to this temporary file

After creating menus, user selects menu-item by pressing the ENTER key, selected choice is redirected to temporary file, Next this menu-item is retrieved from temporary file and following case statement compare the menu-item and takes appropriate step according to selected menu item. As you see, dialog utility allows more powerful user interaction then the older read and echo statement. The only problem with dialog utility is it work slowly.

[Prev](#)

Input (inputbox) using dialog utility

[Home](#)

[Up](#)

[Next](#)

trap command

trap command

Consider following script example:

```
$ cat > testsign
ls -R /
```

Save and run it as

```
$ chmod +x testsign
```

```
$ ./testsign
```

Now if you press `ctrl + c`, while running this script, script get terminated. The `ctrl + c` here work as signal, When such signal occurs its send to all process currently running in your system. Now consider following shell script:

```
$ cat > testsign1
#
# Why to trap signal, version 1
#
Take_input1()
{
    recno=0
    clear
    echo "Appointment Note keeper Application for Linux"
    echo -n "Enter your database file name : "
    read filename
    if [ ! -f $filename ]; then
        echo "Sorry, $filename does not exist, Creating $filename database"
        echo "Appointment Note keeper Application database file" > $filename
    fi
    echo "Data entry start data: `date`" >/tmp/input0.$$
    #
    # Set a infinite loop
    #
    while :
    do
        echo -n "Appointment Title:"
        read na
        echo -n "time :"
        read ti
        echo -n "Any Remark : "
        read remark
        echo -n "Is data okay (y/n) ?"
```

```

    read ans
if [ $ans = y -o $ans = Y ]; then
    recno=`expr $recno + 1`
    echo "$recno. $na $ti $remark" >> /tmp/input0.$$
fi
echo -n "Add next appointment (y/n)?"
read isnext
if [ $isnext = n -o $isnext = N ]; then
    cat /tmp/input0.$$ >> $filename
    rm -f /tmp/input0.$$
    return # terminate loop
fi
done
}
#
#
# Call our user define function : Take_input1
#
Take_input1

```

Save it and run as

```
$ chmod +x testsign1
```

```
$ ./testsign1
```

It first ask you main database file where all appointment of the day is stored, if no such database file found, file is created, after that it open one temporary file in /tmp directory, and puts today's date in that file. Then one infinite loop begins, which ask appointment title, time and remark, if this information is correct its written to temporary file, After that, script asks user , whether he/she wants to add next appointment record, if yes then next record is added , otherwise all records are copied from temporary file to database file and then loop will be terminated. You can view your database file by using cat command. Now problem is that while running this script, if you press CTRL + C, your shell script gets terminated and temporary file are left in /tmp directory. For e.g. try it as follows

```
$/testsign1
```

After given database file name and after adding at least one appointment record to temporary file press CTRL+C, Our script get terminated, and it left temporary file in /tmp directory, you can check this by giving command as follows

```
$ ls /tmp/input*
```

Our script needs to detect such signal (event) when occurs; To achieve this we have to first detect Signal using trap command.

Syntax:

```
trap {commands} {signal number list}
```

Signal Number	When occurs
0	shell exit
1	hangup
2	interrupt (CTRL+C)

3	quit
9	kill (cannot be caught)

To catch signal in above script, put trap statement before calling Take_input1 function as trap del_file 2 .. Here trap command called del_file() when 2 number interrupt (i.e. CTRL+C) occurs. Open above script in editor and modify it so that at the end it will look like as follows:

```
$ vi testsign1
#
# signal is trapped to delete temporary file , version 2
#
del_file()
{
    echo "* * * CTRL + C Trap Occurs (removing temporary file)* * *"
    rm -f /tmp/input0.$$
    exit 1
}

Take_input1()
{
    recno=0
    clear
    echo "Appointment Note keeper Application for Linux"
    echo -n "Enter your database file name : "
    read filename
    if [ ! -f $filename ]; then
        echo "Sorry, $filename does not exist, Creating $filename database"
        echo "Appointment Note keeper Application database file" > $filename
    fi
    echo "Data entry start data: `date`" >/tmp/input0.$$
    #
    # Set a infinite loop
    #
    while :
    do
        echo -n "Appointment Title:"
        read na
        echo -n "time :"
        read ti
        echo -n "Any Remark :"
        read remark
        echo -n "Is data okay (y/n) ?"
        read ans
        if [ $ans = y -o $ans = Y ]; then
            recno=`expr $recno + 1`
        fi
    done
}
```

```

echo "$recno. $na $ti $remark" >> /tmp/input0.$$
fi
echo -n "Add next appointment (y/n)?"
read isnext
if [ $isnext = n -o $isnext = N ]; then
    cat /tmp/input0.$$ >> $filename
    rm -f /tmp/input0.$$
    return # terminate loop
fi
done # end_while
}
#
# Set trap to for CTRL+C interrupt i.e. Install our error handler
# When occurs it first calls del_file() and then exit
#
trap del_file 2
#
# Call our user define function : Take_input1
#
Take_input1

```

Run the script as:

```
$ ./testsign1
```

After giving database file name and after giving appointment title press CTRL+C, Here we have already captured this CTRL + C signal (interrupt), so first our function del_file() is called, in which it gives message as "* * * CTRL + C Trap Occurs (removing temporary file)* * * " and then it remove our temporary file and then exit with exit status 1. Now check /tmp directory as follows

```
$ ls /tmp/input*
```

Now Shell will report no such temporary file exit.

[Prev](#)

User Interface using dialog Utility - Putting it all together

[Home](#)

[Up](#)

[Next](#)

The shift command

The shift Command

The shift command moves the current values stored in the positional parameters (command line args) to the left one position. For example, if the values of the current positional parameters are:

```
$1 = -f $2 = foo $3 = bar
```

and you executed the shift command the resulting positional parameters would be as follows:

```
$1 = foo $2 = bar
```

For e.g. Write the following shell script to clear you idea:

```
$ vi shiftdemo.sh
echo "Current command line args are: \$1=$1, \$2=$2, \$3=$3"
shift
echo "After shift command the args are: \$1=$1, \$2=$2, \$3=$3"
```

Excute above script as follows:

```
$ chmod +x shiftdemo.sh
```

```
$ ./shiftdemo -f foo bar
```

Current command line args are: \$1=-f, \$2=foo, \$3=bar

After shift command the args are: \$1=foo, \$2=bar, \$3=

You can also move the positional parameters over more than one place by specifying a number with the shift command. The following command would shift the positional parameters two places:

```
shift 2
```

But where to use shift command?

You can use shift command to parse the command line (args) option. For example consider the following simple shell script:

```

$ vi convert
while [ "$1" ]
do
    if [ "$1" = "-b" ]; then
        ob="$2"
        case $ob in
            16) basesystem="Hex" ;;
            8) basesystem="Oct" ;;
            2) basesystem="bin" ;;
            *) basesystem="Unknown" ;;
        esac
        shift 2
    elif [ "$1" = "-n" ]
    then
        num="$2"
        shift 2
    else
        echo "Program $0 does not recognize option $1"
        exit 1
    fi
done
output=`echo "obase=$ob;ibase=10; $num;" | bc`
echo "$num Decimal number = $output in $basesystem number
system(base=$ob) "

```

Save and run the above shell script as follows:

```
$ chmod +x convert
```

```
$ ./convert -b 16 -n 500
```

```
500 Decimal number = 1F4 in Hex number system(base=16)
```

```
$ ./convert -b 8 -n 500
```

```
500 Decimal number = 764 in Oct number system(base=8)
```

```
$ ./convert -b 2 -n 500
```

```
500 Decimal number = 111110100 in bin number system(base=2)
```

```
$ ./convert -b 2 -v 500
```

```
Program ./convert does not recognize option -v
```

```
$ ./convert -t 2 -v 500
```

```
Program ./convert does not recognize option -t
```

```
$ ./convert -b 4 -n 500
```

```
500 Decimal number = 13310 in Unknown number system(base=4)
```

```
$ ./convert -n 500 -b 16
```

```
500 Decimal number = 1F4 in Hex number system(base=16)
```

Above script is run in variety of ways. First three sample run converts the number 500 (-n 500) to respectively 1F4 (hexadecimal number i.e. -b 16), 764 (octal number i.e. -b 16) , 111110100 (binary number i.e. -b 16). It use **-n** and **-b** as command line option which means:
-b {base-system i.e. 16,8,2 to which *-n number* to convert}

`-n {Number to convert to -b base-system}`

Fourth and fifth sample run produce the error "*Program ./convert does not recognize option -v*". This is because these two (`-v` & `-t`) are not the valid command line option.

Sixth sample run produced output "*500 Decimal number = 13310 in Unknown number system(base=4)*". Because the base system 4 is unknown to our script.

Last sample run shows that command line options can given different ways i.e. you can use it as follows:

\$./convert -n 500 -b 16

Instead of

\$./convert -b 16 -n 500

All the shell script command can be explained as follows:

Command(s)/Statements	Explanation
<code>while ["\$1"] do</code>	Begins the while loop; continue the while loop as long as script reads the all command line option
<code>if ["\$1" = "-b"]; then ob="\$2"</code>	Now start to parse the command line (args) option using if command our script understands the <code>-b</code> and <code>-n</code> options only all other option are invalid. If option is <code>-b</code> then stores the value of second command line arg to variable <code>ob</code> (i.e. if arg is <code>-b 16</code> then store the 16 to <code>ob</code>)
<code>case \$ob in 16) basesystem="Hex";; 8) basesystem="Oct";; 2) basesystem="bin";;) basesystem="Unknown";; esac</code>	For easy understanding of conversion we store the respective number base systems corresponding string to <code>basesystem</code> variable. If base system is 16 then store the Hex to <code>basesystem</code> and so on. This is done using case statement.
<code>shift 2</code>	Once first two command line options (args) are read, we need next two command line option (args). <code>shift 2</code> will moves the current values stored in the positional parameters (command line args) to the left two position.

<pre>elif ["\$1" = "-n"] then num="\$2" shift 2</pre>	<p>Now check the next command line option and if its -n option then stores the value of second command line arg to variable num (i.e. if arg is -n 500 then store the 500 to num) and shift 2 will moves the current values stored in the positional parameters (command line args) to the left two position.</p>
<pre>else echo "Program \$0 does not recognize option \$1" exit 1 fi</pre>	<p>If command line option is not -n or -b then print the error "<i>Program ./convert does not recognize option xx</i>" on screen and terminates the shell script using <i>exit 1</i> statement.</p>
<pre>done</pre>	<p>End of loop as we read all the valid command line option/args.</p>
<pre>output=`echo "obase=\$ob;ibase=10; \$num;" BC` echo "\$num Decimal number = \$output in \$basesystem number system(base=\$ob)"</pre>	<p>Now convert the given number to given number system using BC Show the converted number on screen.</p>

As you can see shift command can use to parse the command line (args) option. This is useful if you have limited number of command line option. If command line options are too many then this approach works slowly as well as complex to write and maintained. You need to use another shell built in command - getopt. Next section shows the use of getopt command. You still need the shift command in conjunction with getopt for other shell scripting work.

[Prev](#)
trap command

[Home](#)
[Up](#)

[Next](#)
getopts command

getopts command

This command is used to check valid command line argument are passed to script. Usually used in while loop.

Syntax:

```
getopts {optstring} {variable1}
```

getopts is used by shell to parse command line argument.

As defined in man pages:

"*optstring* contains the option letters to be recognized; if a letter is followed by a colon, the option is expected to have an argument, which should be separated from it by white space. Each time it is invoked, *getopts* places the next option in the shell variable *variable1*, When an option requires an argument, *getopts* places that argument into the variable *OPTARG*. On errors *getopts* diagnostic messages are printed when illegal options or missing option arguments are encountered. If an illegal option is seen, *getopts* places ? into *variable1*."

Example:

We have script called *ani* which has syntax as

```
ani -n -a -s -w -d
```

Options: These are optional argument

- n name of animal

- a age of animal

- s sex of animal

- w weight of animal

- d demo values (if any of the above options are used their values are not taken)

Above *ani* script is as follows:

```
$ vi ani
#
# Usage: ani -n -a -s -w -d
#
#
# help_ani() To print help
#
help_ani()
{
  echo "Usage: $0 -n -a -s -w -d"
  echo "Options: These are optional argument"
  echo " -n name of animal"
  echo " -a age of animal"
  echo " -s sex of animal "
```

```
echo " -w weight of animal"
echo " -d demo values (if any of the above options are used "
echo " their values are not taken)"
exit 1
}
#
#Start main procedure
#
#
#Set default value for variable
#
isdef=0
na=Moti
age="2 Months" # may be 60 days, as U like it!
sex=Male
weight=3Kg
#
#if no argument
#
if [ $# -lt 1 ]; then
    help_ani
fi
while getopt n:a:s:w:d opt
do
    case "$opt" in
        n) na="$OPTARG";;
        a) age="$OPTARG";;
        s) sex="$OPTARG";;
        w) weight="$OPTARG";;
        d) isdef=1;;
        \?) help_ani;;
    esac
done
if [ $isdef -eq 0 ]
then
    echo "Animal Name: $na, Age: $age, Sex: $sex, Weight: $weight (user
define mode)"
else
    na="Pluto Dog"
    age=3
    sex=Male
    weight=20kg
    echo "Animal Name: $na, Age: $age, Sex: $sex, Weight: $weight (demo
mode)"
fi
```

Save it and run as follows

```
$ chmod +x ani
```

```
$ ani -n Lassie -a 4 -s Female -w 20Kg
```

```
$ ani -a 4 -s Female -n Lassie -w 20Kg
```

```
$ ani -n Lassie -s Female -w 20Kg -a 4
```

```
$ ani -w 20Kg -s Female -n Lassie -a 4
```

```
$ ani -w 20Kg -s Female
```

```
$ ani -n Lassie -a 4
```

```
$ ani -n Lassie
```

```
$ ani -a 2
```

See because of getopts, we can pass command line argument in different style. Following are invalid options for ani script

```
$ ani -nLassie -a4 -sFemal -w20Kg
```

No space between option and their value.

```
$ ani -nLassie-a4-sFemal-w20Kg
```

```
$ ani -n Lassie -a 4 -s Female -w 20Kg -c Mammal
```

-c is not one of the valid options.

[Prev](#)

The shift command

[Home](#)

[Up](#)

[Next](#)

Essential Utilities for Power User

Introduction

Linux contains powerful utility programs. You can use these utility to

- Locate system information
- For better file management
- To organize your data
- System administration etc

Following section introduce you to some of the essential utilities as well as expression. While programming shell you need to use these essential utilities. Some of these utilities (especially sed & awk) requires understanding of expression. After the quick introduction to utilities, you will learn the expression.

Prepering for Quick Tour of essential utilities

For this part of tutorial create *sname* and *smark* data files as follows (Using text editor of your choice)

Note Each data block is separated from the other by TAB character i.e. while creating the file if you type 11 then press "tab" key, and then write Vivek (as shown in following files):

[sname](#)

<u>Sr.No</u>	<u>Name</u>
11	Vivek
12	Renuka
13	Prakash
14	Ashish
15	Rani

[smark](#)

<u>Sr.No</u>	<u>Mark</u>
11	67
12	55
13	96
14	36
15	67

Selecting portion of a file using cut utility

Suppose from *sname* file you wish to print name of student on-screen, then from shell (Your command prompt i.e. \$) issue command as follows:

```
$cut -f2 sname
```

```
Vivek
```

```
Renuka
```

```
Prakash
```

```
Ashish
```

```
Rani
```

cut utility cuts out selected data from *sname* file. To select Sr.no. field from *sname* give command as follows:

```
$cut -f1 sname
```

```
11
```

```
12
```

```
13
```

```
14
```

```
15
```

Command	Explanation
cut	Name of cut utility
-f1	Using (-f) option, you are specifying the extraction field number. (In this example its 1 i.e. first field)
sname	File which is used by cut utility and which is use as input for cut utility.

You can redirect output of cut utility as follows

```
$cut -f2 sname > /tmp/sn.tmp.$$
```

```
$cut -f2 smark > /tmp/sm.tmp.$$
```

```
$cat /tmp/sn.tmp.$$
```

```
Vivek
```

```
Renuka
```

```
Prakash
```

```
Ashish
```

```
Rani
```

```
$cat /tmp/sm.tmp.$$
```

```
67
```

```
55
```

```
96
```

```
36
```

```
67
```

General Syntax of cut utility:

Syntax:

```
cut -f{field number} {file-name}
```

Use of Cut utility:

Selecting portion of a file.

[Prev](#)

Preparing for Quick Tour of essential utilities

[Home](#)

[Up](#)

[Next](#)

Putting lines together using paste utility

Putting lines together using paste utility

Now enter following command at shell prompt

```
$ paste sname smark
```

```
11 Vivek 11 67
```

```
12 Renuka 12 55
```

```
13 Prakash 13 96
```

```
14 Ashish 14 36
```

```
15 Rani 15 67
```

Paste utility join *textual information together*. To clear your idea try following command at shell prompt:

```
$ paste /tmp/sn.tmp.$$ /tmp/sm.tmp.$$
```

```
Vivek 67
```

```
Renuka 55
```

```
Prakash 96
```

```
Ashish 36
```

```
Rani 67
```

Paste utility is useful to put textual information together located in various files.

General Syntax of paste utility:

Syntax:

```
paste {file1} {file2}
```

Use of paste utility:

Putting lines together.

Can you note down basic difference between cut and paste utility?

The join utility

Now enter following command at shell prompt:

\$join sname smark

11 Vivek 67

12 Renuka 55

13 Prakash 96

14 Ashish 36

15 Rani 67

Here students names are matched with their appropriate marks. How ? join utility uses the Sr.No. field to join to files. Notice that Sr.No. is the first field in both sname and smark file.

General Syntax of join utility:

Syntax:

join {file1} {file2}

Use of join utility:

The join utility joins, lines from separate files.

Note that join will only work, if there is *common field in both file and if values are identical to each other*.

Translateing range of characters using tr utility

Type the following command at shell prompt:

```
$ tr "h2" "3x" < sname
```

```
11 Vivek
```

```
1x Renuka
```

```
13 Prakas3
```

```
14 As3is3
```

```
15 Rani
```

You can clearly see that each occurrence of character 'h' is replace with '3' and '2' with 'x'. tr utility translate specific characters into other specific characters or range of characters into other ranges.

```
h -> 3
```

```
2 -> x
```

Consider following example: (after executing command type text in lower case)

```
$ tr "[a-z]" "[A-Z]"
```

```
hi i am Vivek
```

```
HI I AM VIVEK
```

```
what a magic
```

```
WHAT A MAGIC
```

{Press CTRL + C to terminate.}

Here tr translate range of characters (i.e. small a to z) into other (i.e. to Capital A to Z) ranges.

General Syntax & use of tr utility:

Syntax:

```
tr {pattern-1} {pattern-2}
```

Use of tr utility:

To translate range of characters into other range of characters.

After typing following paragraph, I came to know my mistake that entire paragraph must be in lowercase characters, how to correct this mistake? (Hint - Use tr utility)

```
$ cat > lcommunity.txt
```

```
THIS IS SAMPLE PARAGRAPH
```

```
WRITTEN FOR LINUX COMMUNITY,
```

```
BY VIVEK G GITE (WHO ELSE?)
```

```
OKAY THAT IS OLD STORY.
```

[Prev](#)

The join utility

[Home](#)

[Up](#)

[Next](#)

Data manipulation using awk utility

Data manipulation using awk utility

Before learning more about awk create data file using any text editor or simply vi:

[inventory](#)

```
egg    order  4
cacke  good  10
cheese okay   4
pen    good  12
floppy good   5
```

After crating file issue command

```
$ awk '/good/ { print $3 }' inventory
```

```
10
```

```
12
```

```
5
```

awk utility, select each record from file containing the word "good" and performs the action of printing the third field (Quantity of available goods.). Now try the following and note down its output.

```
$ awk '/good/ { print $1 " " $3 }' inventory
```

General Syntax of awk utility:

Syntax:

```
awk 'pattern action' {file-name}
```

For `$ awk '/good/ { print $3 }' inventory` example,

<code>/good/</code>	Is the pattern used for selecting lines from file.
<code>{print \$3}</code>	This is the action; if pattern found, print on of such action. Here \$3 means third record in selected record. (What \$1 and \$2 mean?)
<code>inventory</code>	File which is used by awk utility which is use as input for awk utility.

Use of awk utility:

To manipulate data.

sed utility - Editing file without using editor

For this part of tutorial create data file as follows

[teormilk](#)

India's milk is good.
tea Red-Lable is good.
tea is better than the coffee.

After creating file give command

```
$ sed '/tea/s//milk/g' teormilk > /tmp/result.tmp.$$
```

```
$ cat /tmp/result.tmp.$$
```

```
India's milk is good.  
milk Red-Lable is good.  
milk is better than the coffee.
```

sed utility is used to find every occurrence of tea and replace it with word milk. sed - Steam line editor which uses 'ex' editors command for editing text files without starting ex. (Cool!, isn't it? no use of text editor to edit anything!!!)

/tea/	Find tea word or select all lines having the word tea
s//milk/	Replace (substitute) the word milk for the tea.
g	Make the changes globally.

Syntax:

```
sed {expression} {file}
```

Use of sed utility: sed is used to edit (text transformation) on given stream i.e a file or may be input from a pipeline.

Removing duplicate lines using uniq utility

Create text file `personame` as follows:

[personame](#)

```
Hello I am vivek
12333
12333
welcome
to
sai computer academy, a'bad.
what still I remeber that name.
oaky! how are u luser?
what still I remeber that name.
```

After creating file, issue following command at shell prompt

```
$ uniq personame
```

```
Hello I am vivek
12333
welcome
to
sai computer academy, a'bad.
what still I remeber that name.
oaky! how are u luser?
what still I remeber that name.
```

Above command prints those lines which are unique. For e.g. our original file contains 12333 twice, so additional copies of 12333 are deleted. But if you examine output of `uniq`, you will notice that 12333 is gone (Duplicate), and "what still I remeber that name" remains as its. Because the `uniq` utility compare only adjacent lines, duplicate lines must be next to each other in the file. To solve this problem you can use command as follows

```
$ sort personame | uniq
```

General Syntax of `uniq` utility:

Syntax:

```
uniq {file-name}
```

[Prev](#)

sed utility - Editing file without using editor

[Home](#)

[Up](#)

[Next](#)

Finding matching pattern using grep utility

Finding matching pattern using grep utility

Create text file as follows:

[demo-file](#)

```
hello world!  
cartoons are good  
especially toon like tom (cat)  
what  
the number one song  
12221  
they love us  
I too
```

After saving file, issue following command,

```
$ grep "too" demofile
```

```
cartoons are good  
especially toon like tom (cat)  
I too
```

grep will locate all lines for the "too" pattern and print all (matched) such line on-screen. grep prints too, as well as cartoons and toon; because grep treat "too" as expression. Expression by grep is read as the letter **t** followed by **o** and so on. So if this expression is found any where on line its printed. grep don't understand words.

Syntax:

```
grep "word-to-find" {file-name}
```

Introduction

In the chapter 5, "Quick Tour of essential utilities", you have seen basic utilities. If you use them with other tools, these utilities are very useful for data processing or for other works. In rest part of tutorial we will learn more about patterns, filters, expressions, and off course sed and awk in depth.

Learning expressions with ex

What does "cat" mean to you ?

One its the word cat, (second cat is an animal! I know 'tom' cat), If same question is asked to computer (not computer but to grep utility) then grep will try to find all occurrence of "cat" word (remember grep read word "cat" as the **c** letter followed by **a** and followed by **t**) including cat, copycat, catalog etc.

Pattern defined as:

"Set of characters (may be words or not) is called pattern."

For e.g. "dog", "celeron", "mouse", "ship" etc are all example of pattern. Pattern can be change from one to another, for e.g. "ship" as "sheep".

Metacharacters defined as:

"If patterns are identified using special characters then such special characters are known as metacharacters".

expressions defined as:

"Combination of pattern and metacharacters is known as expressions (regular expressions)."

Regular expressions are used by different Linux utilities like

- grep
- awk
- sed

So you must know how to construct regular expression. In the next part of LSST you will learn how to construct regular expression using ex editor.

For this part of chapter/tutorial create '[demofile](#)' - text file using any text editor.

Getting started with ex

You can start the ex editor by typing ex at shell prompt:

Syntax:

```
ex {file-name}
```

Example:

```
$ ex demofile
```

The **:** (colon) is ex prompt where you can type ex text editor command or regular expression. Its time to open our demofile, use ex as follows:

```
$ ex demofile
```

```
"demofile" [noeol] 20L, 387C
```

```
Entering Ex mode. Type "visual" to go to Normal mode.
```

```
:
```

As you can see, you will get **:** prompt, here you can type ex command, type **q** and press ENTER key to exit from ex as shown follows: (remember commands are case sensitive)

```
: q
```

```
vivek@ls vivek]$
```

After typing the **q** command you are exit to shell prompt.

Printing text on-screen

First open the our demofile as follows:

```
$ ex demofile
```

```
"demofile" [noeol] 20L, 387C
```

```
Entering Ex mode. Type "visual" to go to Normal mode.
```

Now type 'p' in front of : as follow and press enter

```
:p
```

```
Okay! I will stop.
```

```
:
```

NOTE By default p command will print current line, in our case its the last line of above text file.

Printing lines using range

Now if you want to print 1st line to next 5 line (i.e. 1 to 5 lines) then give command

```
:1,5 p
```

```
Hello World.
```

```
This is vivek from Poona.
```

```
I love linux.
```

```
It is different from all other Os
```

NOTE Here 1,5 is the address. if single number is used (e.g. 5 p) it indicate line number and if two numbers are separated by comma its range of line.

Printing particular line

To print 2nd line from our file give command

```
:2 p
```

```
This is vivek from Poona.
```

Printing entire file on-screen

Give command

```
:1,$ p
```

```
Hello World.
```

```
This is vivek from Poona.
```

```
I love linux.
```

```
It is different from all other Os
```

.....
...
.....

Okay! I will stop.

NOTE Here 1 is 1st line and \$ is the special character of ex which mean last-line character. So 1,\$ means print from 1st line to last-line character (i.e. end of file). Here p stands print.

Printing line number with our text

Give command

```
:set number  
:1,3 p
```

```
1 Hello World.  
2 This is vivek from Poona.  
3
```

NOTE This command prints number next to each line. If you don't want number you can turn off numbers by issuing following command

```
:set nonumber  
:1,3 p
```

```
Hello World.  
This is vivek from Poona.
```

[Prev](#)
Getting started with ex

[Home](#)
[Up](#)

[Next](#)
Deleting lines

Deleting lines

Give command

```
:1, d
```

```
I love linux.
```

NOTE

Here 1 is 1st line and d command indicates deletes (Which deletes the 1st line).

You can even delete range of line by giving command as

```
:1,5 d
```

Copying lines

Give command as follows

```
:1,4 co $
```

```
:1,$ p
```

I love linux.

It is different from all other Os

....

.....

. (DOT) is special command of linux.

Okay! I will stop.

I love linux.

It is different from all other Os

My brother Vikrant also loves linux.

NOTE Here 1,4 means copy 1 to 4 lines; co command stands for copy; \$ is end of file. So it mean copy first four line to end of file. You can delete this line as follows

```
:18,21 d
```

```
:1,$ p
```

```
:1,$ p
```

I love linux.

It is different from all other Os

My brother Vikrant also loves linux.

He currently lerarns linux.

Linux is coool.

Linux is now 10 years old.

Next year linux will be 11 year old.

Rani my sister never uses Linux

She only loves to play games and nothing else.

Do you know?

. (DOT) is special command of linux.

Okay! I will stop.

[Prev](#)

Deleting lines

[Home](#)

[Up](#)

[Next](#)

Searching the words

Searching the words

(a) Give following command

```
:/linux/ p
```

I love linux.

Note In ex you can specify address (line) using number for various operation. This is useful if you know the line number in advance, but if you don't know line number, then you can use *contextual address* to print line on-screen. In above example */linux/* is *contextual address* which is constructed by surrounding a regular expression with two slashes. And p is print command of ex.

Try following and note down difference (Hint - Watch p is missing)

```
:/Linux/
```

(b) Give following command

```
:g/linux/ p
```

I love linux.

My brother Vikrant also loves linux.

He currently learns linux.

Next year linux will be 11 year old.

. (DOT) is special command of linux.

In previous example (*:/linux/ p*) only one line is printed. If you want to print all occurrence of the word "linux" then you have to use g, which mean global line address. This instruct ex to find all occurrence of pattern. Try following

```
:1,$ /Linux/ p
```

Which give the same result. It means g stands for 1,\$.

Saving the file in ex

Give command

```
:w
```

"demofile" 20L, 386C written

w command will save the file.

Quitting the ex

Give command

```
:q
```

q command quits from ex and you are return to shell prompt.

Note use wq command to do save and exit from ex.

[Prev](#)

Coping lines

[Home](#)

[Up](#)

[Next](#)

Find and Replace (Substituting regular expression)

Find and Replace (Substituting regular expression)

Give command as follows

```
:8 p
```

He currently lerarns linux.

```
:8 s/lerarns/learn/
```

```
:p
```

He currently learn linux.

Note Using above command, you are substituting the word "learn" for the word "lerarns".

Above command can be explained as follows:

Command	Explanation
8	Goto line 8, address of line.
s	Substitute
/lerarns/	Target pattern
learn/	If target pattern found substitute the expression (i.e. learn/)

Considered the following command:

```
:1,$ s/Linux/Unix/
```

Rani my sister never uses Unix

```
:1,$ p
```

Hello World.

This is vivek from Poona.

....

..

.....

. (DOT) is special command of linux.

Okay! I will stop.

Using above command, you are substituting all lines i.e. s command will find all of the address line for the pattern "Linux" and if pattern "Linux" found substitute pattern "Unix".

Command	Explanation
:1,\$	Substitute for all line
s	Substitute

/Linux/	Target pattern
Unix/	If target pattern found substitute the expression (i.e. Unix/)

Even you can also use **contextual address** as follows

:/sister/ p

Rani my sister never uses Unix

:g /sister/ s/never/always/

:p

Rani my sister always uses Unix

Above command will first find the line containing pattern "sister" if found then it will substitute the pattern "always" for the pattern "never" (It mean find the line containing the word sister, on that line find the word never and replace it with word always.)

Try the following and watch the output very carefully.

:g /Unix/ s/Unix/Linux

3 substitutions on 3 lines

Above command finds all line containing the regular expression "Unix", then substitute "Linux" for all occurrences of "Unix". Note that above command can be also written as follows

:g /Unix/ s//Linux

Here // is replace by the last pattern/regular expression i.e. Unix. Its shortcut. Now try the following

:g /Linux/ s//UNIX/

3 substitutions on 3 lines

:g/Linux/p

Linux is coool.

Linux is now 10 years old.

Rani my sister always uses Linux

:g /Linux/ s//UNIX/

3 substitutions on 3 lines

:g/UNIX/p

UNIX is coool.

UNIX is now 10 years old.

Rani my sister always uses UNIX

By default substitute command only substitute first occurrence of a pattern on a line. Let's take another example, give command

:/brother/p

My brother Vikrant also loves linux who also loves unix.

Now in above line "also" word is occurred twice, give the following substitute command

:g/brother/ s/also/XYZ/

:/brother/p

My brother Vikrant XYZ loves linux who also loves unix.

Make sure next time it works

```
:g/brother/ s/XYZ/also/
```

Note that "also" is only once substituted. If you want to **s** command to work with all occurrences of pattern within a address line give command as follows:

```
:g/brother/ s/also/XYZ/g
```

```
:p
```

My brother Vikrant XYZ loves linux who XYZ loves unix.

```
:g/brother/ s/XYZ/also/g
```

```
:p
```

My brother Vikrant also loves linux who also loves unix.

The **g** option at the end instruct **s** command to perform replacement on all occurrences of the target pattern within a address line.

[Prev](#)

[Home](#)

[Next](#)

Searching the words

[Up](#)

Replacing word with confirmation from
user

Replacing word with confirmation from user

Give command as follows

```
:g/Linux/ s//UNIX/gc
```

After giving this command ex will ask you question like - *replace with UNIX (y/n/a/q/^E/^Y)?*

Type **y** to replace the word or **n** to not replace or **a** to replace all occurrence of word.

Finding words

Command like

```
:g/the/p
```

It is different from all other Os

My brother Vikrant also loves linux who also loves unix.

Will find word like theater, the, brother, other etc. What if you want to just find the word like "the" ? To find the word (Let's say Linux) you can give command like

```
:/^<Linux\>
```

Linux is coool.

```
:g/^<Linux\>/p
```

Linux is coool.

Linux is now 10 years old.

Rani my sister never uses Linux

The symbol \< and \> respectively match the empty string at the beginning and end of the word. To find the line which contain Linux pattern at the beginning give command

```
:/^Linux
```

Linux is coool.

As you know \$ is end of line character, the ^ (caret) match beginning of line. To find all occurrence of pattern "Linux" at the beginning of line give command

```
:g/^Linux
```

Linux is coool.

Linux is now 10 years old.

And if you want to find "Linux" at the end of line then give command

```
:/Linux $
```

Rani my sister never uses Linux

Following command will find empty line:

```
:/^$
```

To find all blank line give command:

```
:g/^$
```

To view entire file without blank line you can use command as follows:

```
:g/[^^$]
```

Hello World.

This is vivek from Poona.

I love linux.

It is different from all other Os

My brother Vikrant also loves linux who also loves unix.

He currently learn linux.

Linux is coool.

Linux is now 10 years old.

Next year linux will be 11 year old.

Rani my sister never uses Linux

She only loves to play games and nothing else.

Do you know?

. (DOT) is special command of linux.

Okay! I will stop.

Command	Explanation
g	All occurrence
/[^	[^] This means not
/^\$	Empty line, Combination of ^ and \$.

To delete all blank line you can give command as follows

:g/^\$/d

Okay! I will stop.

:1,\$ p

Hello World.

This is vivek from Poona.

I love linux.

It is different from all other Os

My brother Vikrant also loves linux who also loves unix.

He currently learn linux.

Linux is coool.

Linux is now 10 years old.

Next year linux will be 11 year old.

Rani my sister never uses Linux

She only loves to play games and nothing else.

Do you know?

. (DOT) is special command of linux.

Okay! I will stop.

Try u command to undo, to undo what you have done it, give it as follows:

:u

:1,\$ p

Hello World.

This is vivek from Poona.

....

...

....

...

....

...

....

...

....

...

....

...

....

...

....

Okay! I will stop.

[Prev](#)

Replacing word with confirmation from user

[Home](#)

[Up](#)

[Next](#)

Using range of characters in regular expressions

Using range of characters in regular expressions

Try the following command

```
:g/Linux/p
```

Linux is coool.

Linux is now 10 years old.

Rani my sister never uses Linux

This will find only "Linux" and not the "linux", to overcome this problem try as follows

```
:g/[Ll]inux/p
```

I love linux.

My brother Vikrant also loves linux who also loves unix.

He currently learn linux.

Linux is coool.

Linux is now 10 years old.

Next year linux will be 11 year old.

Rani my sister never uses Linux

. (DOT) is special command of linux.

Here a list of characters enclosed by [and], which matches any single character in that range. if the first character of list is ^, then it matches any character not in the list. In above example [Ll], will try to match L or l with rest of pattern. Let's see another example. Suppose you want to match single digit character in range you can give command as follows

```
:/[0123456789]
```

Even you can try it as follows

```
:g/[0-9]
```

Linux is now 10 years old.

Next year linux will be 11 year old.

Here range of digit is specified by giving first digit (0-zero) and last digit (9), separated by hyphen. You can try [a-z] for lowercase character, [A-Z] for uppercase character. Not just this, there are certain named classes of characters which are predefined. They are as follows:

Predefined classes of characters	Meaning
[:alnum:]	Letters and Digits (A to Z or a to z or 0 to 9)
[:alpha:]	Letters A to Z or a to z
[:cntrl:]	Delete character or ordinary control character (0x7F or 0x00 to 0x1F)

[digit:]	Digit (0 to 9)
[graph:]	Printing character, like print, except that a space character is excluded
[lower:]	Lowercase letter (a to z)
[print:]	Printing character (0x20 to 0x7E)
[punct:]	Punctuation character (ctrl or space)
[space:]	Space, tab, carriage return, new line, vertical tab, or form feed (0x09 to 0x0D, 0x20)
[upper:]	Uppercase letter (A to Z)
[xdigit:]	Hexadecimal digit (0 to 9, A to F, a to f)

For e.g. To find digit or alphabet (Upper as well as lower) you will write

```
:[0-9A-Za-Z]
```

Instead of writing such command you could easily use predefined classes or range as follows

```
:[[:alnum:]]
```

The . (dot) matches any single character.

For e.g. Type following command

```
:g\<<.o\>
```

She only loves to play games and nothing else.

Do you know?

This will include lo(ves), Do, no(thing) etc.

*** Matches the zero or more times**

For e.g. Type following command

```
:g/L*
```

Hello World.

This is vivek from Poona.

....

....

```
:g/Li*
```

Linux is coool.

Linux is now 10 years old.

Rani my sister never uses Linux

```
:g/c.*and
```

. (DOT) is special command of linux.

Here first **c** character is matched, then any single character (.) followed by n number of single character (1 or 100 times even) and finally ends with and. This can found different word as follows command or catand etc.

In the regular expression metacharacters such as . (DOT) or * loose their special meaning if we use as \<. or \<*. The backslash removes the special meaning of such meatcharacters and you can use them as ordinary characters. For e.g. If u want to search . (DOT) character at the beginning of line, then you can't

use command as follows

:g/^\.

Hello World.

This is vivek from Poona.

....

..

...

. (DOT) is special command of linux.

Okay! I will stop.

Instead of that use

:g/^\.

. (DOT) is special command of linux.

[Prev](#)

Finding words

[Home](#)

[Up](#)

[Next](#)

Using & as Special replacement character

Using & as Special replacement character

Try the following command:

```
:1,$ s/Linux/&-Unix/p
```

3 substitutions on 3 lines

Rani my sister never uses Linux-Unix

```
:g/Linux-Unix/p
```

Linux-Unix is coool.

Linux-Unix is now 10 years old.

Rani my sister never uses Linux-Unix

This command will replace, target pattern "Linux" with "Linux-Unix". & before - Unix means use "last pattern found" with given pattern, So here last pattern found is "Linux" which is used with given -Unix pattern (Finally constructing "Linux-Unix" substitute for "Linux").

Can you guess the output of this command?

```
:1,$ s/Linux-Unix/&Linux/p
```

Converting lowercase character to uppercase

Try the following command

```
:1,$ s/[a-z]/\u &/g
```

Above command can be explained as follows:

Command	Explanation
1,\$	Line Address location is all i.e. find all lines for following pattern
s	Substitute command
/[a-z]/	Find all lowercase letter - Target
\u&/	Substitute to Uppercase. \u& means substitute last patter (&) matched with its UPPERCASE replacement (\u) <u>Note:</u> Use \l (small L) for lowercase character.
g	Global replacement

Can you guess the output of following command?

```
:1,$ s/[A-Z]/\l&/g
```

Congratulation, for successfully completion of this tutorial of regular expressions.

I hope so you have learn lot from this. To master the expression you have to do lot of practice. This tutorial is very important to continue with rest of tutorial and to become power user of Linux. Impress your friends with such expressions. Can you guess what last expression do?

```
:1,$ s/^_ _*$//
```

Note : `_ _` indicates two black space.

Introduction : awk - Revisited

awk utility is powerful data manipulation/scripting programming language (In fact based on the C programming Language). Use awk to handle complex task such as calculation, database handling, report creation etc.

General Syntax of awk:

Syntax:

```
awk -f {awk program file} filename
```

awk Program contains are something as follows:

```
Pattern {
    action 1
    action 2
    action N
}
```

awk reads the input from given file (or from stdin also) one line at a time, then each line is compared with pattern. If pattern is match for each line then given action is taken. Pattern can be regular expressions. Following is the summery of common awk metacharacters:

Metacharacter	Meaning
. (Dot)	Match any character
*	Match zero or more character
^	Match beginning of line
\$	Match end of line
\	Escape character following
[]	List
{ }	Match range of instance
+	Match one more preceding
?	Match zero or one preceding
	Separate choices to match

Getting Starting with awk

Consider following text database file

Sr.No	Product	Qty	Unit Price
1	Pen	5	20.00
2	Rubber	10	2.00
3	Pencil	3	3.50
4	Cock	2	45.50

In above file fields are Sr.No,Product,Qty,Unit Price. Field is the smallest element of any record. Each fields has its own attributes. For e.g. Take Qty. field. Qty. fields attribute is its numerical (Can contain only numerical data). Collection of fields is know as record. So

1. Pen 5 20.00 ----> Is a Record.

Collection of record is know as *database file*. In above text database file each field is separated using space (or tab character) and record is separated using new-line character (i.e. each record is finished at the end of line). In the awk, fields are access using special variable. For e.g. In above database \$1, \$2, \$3, \$4 respectively represents Sr.No, Product, Qty, Unit Price fields. (Don't confuse \$1,\$2 etc with command line arguments of shell script)

For this part of tutorial create text datafile [inven](#) (Shown as above). Now enter following simple awk program/command at shell prompt:

```
$ awk '{ print $1 $2 "--> Rs." $3 * $4 }' inven
```

1.Pen--> Rs.100

2.Pencil--> Rs.20

3.Rubber--> Rs.10.5

4.Cock--> Rs.91

Above awk program/command can be explained as follows:

awk program statement	Explanation
<pre>'{ print \$1 \$2 "--> Rs." \$3 * \$4 }'</pre>	print command is used to print contains of variables or text enclose in " text ". Here \$1, \$2, \$3,\$4 are all the special variable. \$1, \$2, etc all of the variable contains value of field. Finally we can directly do the calculation using \$3 * \$4 i.e. multiplication of third and fourth field in database. Note that "--> Rs." is string which is printed as its.

Note \$1,\$2 etc (in awk) also know as predefined variable and can assign any value found in field.

Type following awk program at shell prompt,


```
$ awk '{ print $2 }' inven
```

```
Pen
```

```
Pencil
```

```
Rubber
```

```
Cock
```

awk prints second field from file. Same way if you want to print second and fourth field from file then give following command:

```
$awk '{ print $2 $4}' inven
```

```
Pen20.00
```

```
Pencil2.00
```

```
Rubber3.50
```

```
Cock45.50
```

\$0 is special variable of awk , which print entire record, you can verify this by issuing following awk command:

```
$ awk '{ print $0 }' inven
```

```
1. Pen 5 20.00
```

```
2. Pencil 10 2.00
```

```
3. Rubber 3 3.50
```

```
4. Cock 2 45.50
```

You can also create awk command (program) file as follows:

```
$ cat > prn_pen
/Pen/ { print $3 }
```

And then you can execute or run above "prn_pen" awk command file as follows

```
$ awk -f prn_pen inven
```

```
5
```

```
10
```

In above awk program `/Pen/` is the search pattern, if this pattern is found on line (or record) then print the third field of record.

`{ print $3 }` is called Action. On shell prompt , `$ awk -f prn_pen inven` , `-f` option instruct awk, to read its command from given file, `inven` is the name of database file which is taken as input for awk.

Now create following awk program as follows:

```
$cat > comp_inv
3 > 5 { print $0 }
```

Run it as follows:

```
$ awk -f comp_inv inven
```

```
2. Pencil 10 2.00
```

Here third field of database is compared with 5, this the pattern. If this pattern found on any line database, then entire record is printed.

[Prev](#)

awk Revisited

[Home](#)

[Up](#)

[Next](#)

Predefined variable of awk

Predefined variable of awk

Our next example talks more about predefined variable of awk. Create awk file as follows:

```
$cat > def_var
{
print "Printing Rec. #" NR "(" $0 "),And # of field for this record is
" NF
}
```

Run it as follows.

\$awk -f def_var inven

Printing Rec. #1(1. Pen 5 20.00),And # of field for this record is 4

Printing Rec. #2(2. Pencil 10 2.00),And # of field for this record is 4

Printing Rec. #3(3. Rubber 3 3.50),And # of field for this record is 4

Printing Rec. #4(4. Cock 2 45.50),And # of field for this record is 4

NR and *NF* are predefined variables of awk which means Number of input Record, Number of Fields in input record respectively. In above example *NR* is changed as our input record changes, and *NF* is constant as there are only 4 field per record. Following table shows list of such built in awk variables.

awk Variable	Meaning
FILENAME	Name of current input file
RS	Input record separator character (Default is new line)
OFS	Output field separator string (Blank is default)
ORS	Output record separator string (Default is new line)
NF	Number of input record
NR	Number of fields in input record
OFMT	Output format of number
FS	Field separator character (Blank & tab is default)

Doing arithmetic with awk

You can easily, do the arithmetic with awk as follows

```
$ cat > math
{
  print $1 " + " $2 " = " $1 + $2
  print $1 " - " $2 " = " $1 - $2
  print $1 " / " $2 " = " $1 / $2
  print $1 " x " $2 " = " $1 * $2
  print $1 " mod " $2 " = " $1 % $2
}
```

Run the awk program as follows:

```
$ awk -f math
```

```
20 3
20 + 3 = 23
20 - 3 = 17
20 / 3 = 6.66667
20 x 3 = 60
20 mod 3 = 2
```

(Press CTRL + D to terminate)

In above program print `$1 " + " $2 " = " $1 + $2`, statement is used for addition purpose. Here `$1 + $2`, means add (+) first field with second field. Same way you can do - (subtraction), * (Multiplication), / (Division), % (modular use to find remainder of division operation).

User Defined variables in awk

You can also define your own variable in awk program, as follows:

```
$ cat > math1
{
no1 = $1
no2 = $2
ans = $1 + $2
print no1 " + " no2 " = " ans
}
```

Run the program as follows

```
$ awk -f math1
```

```
1 5
```

```
1 + 5 = 6
```

In the above program, no1, no2, ans all are user defined variables. Value of first and second field are assigned to no1, no2 variable respectively and the addition to ans variable. Value of variable can be printed using print statement as, print no1 " + " no2 " = " ans. Note that print statement prints whatever enclosed in double quotes (" text ") as it is. If string is not enclosed in double quotes its treated as variable. Also above two program takes input from stdin (Keyboard) instead of file.

Now try the following awk program and note down its output.

```
$ cat > bill
{
total = $3 * $4
recno = $1
item = $2
print recno item " Rs." total
}
```

Run it as

```
$ awk -f bill inven
```

```
1.Pen Rs.100
```

```
2.Pencil Rs.20
```

```
3.Rubber Rs.10.5
```

```
4.Cock Rs.91
```

Here we are printing the total price of each product (By multiplying third field with fourth field). Following program prints total price of each product as well as the Grand total of all product in the bracket.

```
$ cat > bill1
{
total = $3 * $4
recno = $1
item = $2
gtotal = gtotal + total
print recno item " Rs." total " [Total Rs." gtotal " ] "
}
```

Run the above awk program as follows:

```
$ awk -f bill1 inven
```

```
1.Pen Rs.100 [Total Rs.100]
2.Pencil Rs.20 [Total Rs.120]
3.Rubber Rs.10.5 [Total Rs.130.5]
4.Cock Rs.91 [Total Rs.221.5]
```

In this program, gtotal variable holds the grand total. It adds the total of each product as `gtotal = gtotal + total`. Finally this total is printed with each record in the bracket. But there is one problem with our script, Grand total mostly printed at the end of all record. To solve this problem we have to use special BEGIN and END Patterns of awk. First take the example,

```
$ cat > bill2
BEGIN {
    print "-----"
    print "Bill for the 4-March-2001. "
    print "By Vivek G Gite. "
    print "-----"
}

{
    total = $3 * $4
    recno = $1
    item = $2
    gtotal += total
    print recno item " Rs." total
}

END {
    print "-----"
    print "Total Rs." gtotal
    print "===== "
}
```

Run it as

```
$awk -f bill2 inven
```

```
-----
Bill for the 4-March-2001.
```

By Vivek G Gite.

```
-----
1.Pen Rs.100
2.Pencil Rs.20
3.Rubber Rs.10.5
4.Cock Rs.91
-----
```

```
Total Rs.221.5
=====
```

Now the grand total is printed at the end. In above program BEGIN and END patters are used. BEGIN instruct awk, that perform BEGIN actions before the first line (Record) has been read from database file. Use BEGIN pattern to set value of variables, to print heading for report etc. General syntax of BEGIN is as follows

Syntax:

```
BEGIN {
    action 1
    action 2
    action N
}
```

END instruct awk, that perform END actions after reading all lines (RECORD) from the database file. General syntax of END is as follows:

```
END {
    action 1
    action 2
    action N
}
```

In our example, BEGIN is used to print heading and END is used print grand total.

[Prev](#)

Doing arithmetic with awk

[Home](#)

[Up](#)

[Next](#)

Use of printf statement

Use of printf statement

Next example shows the use of special printf statement

```
$ cat > bill3
BEGIN {
    printf "Bill for the 4-March-2001.\n"
    printf "By Vivek G Gite.\n"
    printf "-----\n"
}

{
    total = $3 * $4
    recno = $1
    item = $2
    gtotal += total
    printf "%d %s Rs.%f\n", recno, item, total
    #printf "%2d %-10s Rs.%7.2f\n", recno, item, total
}

END {
    printf "-----\n"
    printf "Total Rs. %f\n" ,gtotal
    #printf "\tTotal Rs. %7.2f\n" ,gtotal
    printf "=====\n"
}
```

Run it as follows:

```
$ awk -f bill3 inven
```

```
Bill for the 4-March-2001.
```

```
By Vivek G Gite.
```

```
-----
1 Pen Rs.100.000000
2 Pencil Rs.20.000000
3 Rubber Rs.10.500000
4 Cock Rs.91.000000
-----
Total Rs. 221.500000
=====
```

In above example printf statement is used to print formatted output of the variables or text. General syntax of printf as follows:

Syntax:

```
printf "format" ,var1, var2, var N
```

If you just want to print any text using printf as follows

```
printf "Hello"
printf "Hello World\n"
```

In last example `\n` is used to print new line. Its Part of escape sequence following may be also used:

```
\t for tab
\a Alert or bell
\" Print double quote etc
```

For e.g. **printf "\nAn apple a day, keeps away\t\t\tDoctor\n\a\a"**

It will print text on new line as :

An apple a day, keeps away Doctor

Notice that twice the sound of bell is produced by `\a\a`. To print the value of decimal number use `%d` as format specification code followed by the variable name. For e.g. `printf "%d" , no1`

It will print the value of `no1`. Following table shows such common format specification code:

Format Specification Code	Meaning	Example
<code>%c</code>	Character	{ isminor = "y" printf "%c" , isminor }
<code>%d</code>	Decimal number such as 10,-5 etc	{ n = 10 printf "%d",n }
<code>%x</code>	Hexadecimal number such as 0xA, 0xffff etc	{ n = 10 printf "%x",n }
<code>%s</code>	String such as "vivek", "Good buy"	{ str1 = "Welcome to Linux!" printf "%s", str1 printf "%s", "Can print ?" }

To run above example simply create any awk program file as follows

```
$ cat > p_demo
BEGIN {
n = 10
printf "%d", n
printf "\nAn apple a day, keeps away\t\t\tDoctor\n\a\a"
}
```

Run it as

```
$ awk -f p_demo
```

```
10
```

```
An apple a day, keeps away Doctor
```

Write awk program to test format specification code. According to your choice.

[Prev](#)

User Defined variables in awk

[Home](#)

[Up](#)

[Next](#)

Use of Format Specification Code

Use of Format Specification Code

```

$ cat > bill4
BEGIN {
    printf "Bill for the 4-March-2001.\n"
    printf "By Vivek G Gite.\n"
    printf "-----\n"
}

{
    total = $3 * $4
    recno = $1
    item = $2
    gtotal += total
    printf "%2d %-10s Rs.%7.2f\n", recno, item, total
}

END {
    printf "-----\n"
    printf "\tTotal Rs. %6.2f\n" ,gtotal
    printf "=====\n"
}

```

Run it as

```
$ awk -f bill4 inven
```

Bill for the 4-March-2001.

By Vivek G Gite.

```

-----
1 Pen    Rs. 100.00
2 Pencil Rs. 20.00
3 Rubber Rs. 10.50
4 Cock   Rs. 91.00
-----
Total    Rs. 221.50
=====

```

From the above output you can clearly see that printf can format the output. Let's try to understand formatting of printf statement. For e.g. %2d, number between % and d, tells the printf that assign 2 spaces for value. Same way if you write following awk program ,

```
$ cat > prf_demo
{
na = $1
printf "|%s|", na
printf "|%10s|", na
printf "|%-10s|", na
}
```

Run it as follows (and type the **God**)

\$ awk -f prf_demo

God

```
/God/
/  God/
/God  /
```

(press CTRL + D to terminate)

printf " %s ", na	Print God as its
printf " %10s ", na	Print God Word as Right justified.
printf " %-10s ", na	Print God Word as left justified. (- means left justified)

Same technique is used in our bill4 awk program to print formatted output. Also the statement like `gtotal += total`, which is equivalent to `gtotal = gtotal + total`. Here `+=` is called assignment operator. You can use following assignment operator:

Assignment operator	Use for	Example	Equivalent to
<code>+=</code>	Assign the result of addition	<code>a += 10</code> <code>d += c</code>	<code>a = a + 10</code> <code>a = a + c</code>
<code>-=</code>	Assign the result of subtraction	<code>a -= 10</code> <code>d -= c</code>	<code>a = a - 10</code> <code>a = a - c</code>
<code>*=</code>	Assign the result of multiplication	<code>a *= 10</code> <code>d *= c</code>	<code>a = a * 10</code> <code>a = a * c</code>
<code>%=</code>	Assign the result of modulo	<code>a %= 10</code> <code>d %= c</code>	<code>a = a % 10</code> <code>a = a % c</code>

if condition in awk

General syntax of if condition is as follows:

Syntax:

```
if ( condition )
{
    Statement 1
    Statement 2
    Statement N
    if condition is TRUE
}
else
{
    Statement 1
    Statement 2
    Statement N
    if condition is FALSE
}
```

Above if syntax is selfexplontary, now lets move to next awk program

```
$ awk > math2
BEGIN {
    myprompt = "(To Stop press CTRL+D) > "
    printf "Welcome to MyAddtion calculation awk program v0.1\n"
    printf "%s" ,myprompt
}

{
no1 = $1
op = $2
no2 = $3
ans = 0

if ( op == "+" )
{
    ans = $1 + $3
    printf "%d %c %d = %d\n" ,no1,op,no2,ans
    printf "%s" ,myprompt
}
}
```

```

else
{
    printf "Opps!Error I only know how to add.\nSyntax: number1 +
number2\n"
    printf "%s" ,myprompt
}
}

END {
    printf "\nGoodbuy %s\n" , ENVIRON["USER"]
}

```

Run it as follows (Give input as **5 + 2** and **3 - 1** which is shown in bold words)

\$awk -f math2

Welcome to MyAddtion calculation awk program v0.1

(To Stop press CTRL+D) > **5 + 2**

5 + 2 = 7

(To Stop press CTRL+D) > **3 - 1**

Opps!Error I only know how to add.

Syntax: number1 + number2

(To Stop press CTRL+D) >

Goodbuy vivek

In the above program various, new concept are introduce so lets try to understand them step by step

BEGIN {	Start of BEGIN Pattern
myprompt = "(To Stop press CTRL+D) > "	Define user defined variable
printf "Welcome to MyAddtion calculation awk program v0.1\n" printf "%s" ,myprompt	Print welcome message and value of myprompt variable.
}	End of BEGIN Pattern
{	Now start to process input
no1 = \$1 op = \$2 no2 = \$3 ans = 0	Assign first, second, third, variables value to no1 , op , no2 variables respectively

<pre>if (op == "+") { ans = no1 + no2 printf "%d %c %d = %d\n" ,no1,op,no2,ans printf "%s" ,myprompt } else { printf "Opps!Error I only know how to add.\nSyntax:number1+ number2\n" printf "%s" ,myprompt } }</pre>	<p>If command is used for decision making in awk program. Here if value of variable op is "+" then addition is done and result is printed on screen, else error message is shown on screen.</p>
<pre>}</pre>	<p>Stop all inputted lines are process.</p>
<pre>END { printf "\nGoodbuy %s\n" , ENVIRON["USER"] }</pre>	<p>END patterns start here. Which says currently log on user Goodbuy.</p>

ENVIRON is the one of the predefined system variable that is array. Array is made up of different element. ENVIRON array is also made of elements. It allows you to access system variable (or variable in your environment). Give set command at shell prompt to see list of your environment variable. You can use variable name to reference any element in this array. For e.g. If you want to print your home directory you can write printf as follows:

```
printf "%s is my sweet home", ENVIRON["HOME"]
```

[Prev](#)

Use of Format Specification Code

[Home](#)

[Up](#)

[Next](#)

Loops in awk

Loops in awk

For loop and [while loop](#) are used for looping purpose in awk.

Syntax of for loop

Syntax:

```
for (expr1; condition; expr2)
```

```
{  
    Statement 1  
    Statement 2  
    Statement N  
}
```

Statement(s) are executed repeatedly UNTIL the condition is true. BEFORE the first iteration, expr1 is evaluated. This is usually used to initialize variables for the loop. AFTER each iteration of the loop, expr2 is evaluated. This is usually used to increment a loop counter.

Example:

```
$ cat > while01.awk  
BEGIN{  
    printf "Press ENTER to continue with for loop example from LSST  
v1.05r3\n"  
}  
{  
sum = 0  
i = 1  
for (i=1; i<=10; i++)  
{  
    sum += i; # sum = sum + i  
}  
printf "Sum for 1 to 10 numbers = %d \nGoodbuy!\n\n", sum  
exit 1  
}
```

Run it as follows:

```
$ awk -f while01.awk
```

Press ENTER to continue with for loop example from LSST v1.05r3

Sum for 1 to 10 numbers = 55

Goodbuy

Above for loops prints the sum of all numbers between 1 to 10, it does use very simple for loop to achieve this. It take number from 1 to 10 using i variable and add it to sum variable as sum = previous sum + current number (i.e. i).

Consider one more example of for loop:

```
$ cat > for_loop
BEGIN {
    printf "To test for loop\n"
    printf "Press CTRL + C to stop\n"
}
{
    for(i=0;i<NF;i++)
    {
        printf "Welcome %s, %d times.\n" ,ENVIRON["USER"], i
    }
}
```

Run it as (and give input as **Welcome to Linux!**)

```
$ awk -f for_loop
```

To test for loop

Press CTRL + C to Stop

Welcome to Linux!

Welcome vivek, 0 times.

Welcome vivek, 1 times.

Welcome vivek, 2 times.

Program uses for loop as follows:

<code>for(i=0;i<NF;i++)</code>	Set the value of i to 0 (Zero); Continue as long as value of i is less than NF (Remember NF is built in variable, which mean Number of Fields in record); increment i by 1 (i++)
<code>printf "Welcome %s, %d times.\n",ENVIRON["USER"], i</code>	Print "Welcome...." message, with user name who is currently log on and value of i.

Here `i++`, is equivalent to `i = i + 1` statement. `++` is increment operator which increase the value of variable by one and `--` is decrement operator which decrease the value of variable by one. Don't try `i+++`, to increase the value of i by 2 (since `+++` is not valid operator), instead try `i+= 2`.

You can use while loop as follows:

Syntax:

```
while (condition)
```

```
{
```

```
    statement1
```

```
    statement2
```

```
    statementN
```

```
    Continue as long as given condition is TRUE
```

```
}
```

While loop will continue as long as given condition is TRUE. To understand the while loop lets write

one more awk script:

```
$ cat > while\_loop
{
no = $1
remn = 0
while ( no > 1 )
{
    remn = no % 10
    no /= 10
    printf "%d" ,remn
}
printf "\nNext number please (CTRL+D to stop):";
}
```

Run it as

\$awk -f while_loop

654

456

Next number please(CTRL+D to stop):587

785

Next number please(CTRL+D to stop):

Here user enters the number 654 which is printed in reverse order i.e. 456. Above program can be explained as follows:

no = \$1	Set the first fields (\$1) value to no.
remn = 0	Set remn variable to zero
{	Start the while loop
while (no > 1)	Continue the loop as long as value of no is greater than one
remn = no % 10	Find the remainder of no variable, and assign result to remn variable.
no /= 10	Divide the no by 10 and store result to no variable.
print "%d", remn	Print the remn (remainder) variables value.
}	End of while loop, since condition (no>1) is not true i.e false condition..
printf "\nNext number please (CTRL+D to stop):";	Prompt for next number

[Prev](#)

if condition in awk

[Home](#)

[Up](#)

[Next](#)

Real life example in awk

Real life example in awk

Before learning more features of awk its time to see some real life example in awk.

Our first Example

I would like to read name of all files from the file and copy them to given destination directory. For e.g. The file [filelist.conf](#); looks something as follows:

```
/home/vivek/awks/temp/file1  /home/vivek/final
/home/vivek/awks/temp/file2  /home/vivek/final
/home/vivek/awks/temp/file3  /home/vivek/final
/home/vivek/awks/temp/file4  /home/vivek/final
```

In above file first field (\$1) is the name of file that I would like to copy to the given destination directory (\$2 - second field) i.e. copy /home/vivek/awks/temp/file1 file to /home/vivek/final directory. For this purpose write the awk program as follows:

```
$ cat > temp2final.awk
#
#temp2final.awk
#Linux Shell Scripting Tutorial v1.05, March 2001
#Author: Vivek G Gite
#
BEGIN{
}

#
# main logic is here
#
{
    sfile = $1
    dfile = $2
    cpcmd = "cp " $1 " " $2
    printf "Coping %s to %s\n",sfile,dfile
    system(cpcmd)
}

#
# End action, if any, e.g. clean ups
#
END{
```

}

Run it as follows:

```
$ awk -f temp2final.awk filelist.conf
```

Above awk Program can be explained as follows:

sfile = \$1	Set source file path i.e. first field (\$1) from the file filelist.conf
dfile = \$2	Set source file path i.e. second field (\$2) from the file filelist.conf
cpcmd = "cp " \$1 " " \$2	Use your normal cp command for copy file from source to destination. Here cpcmd , variable is used to construct cp command.
printf "Coping %s to %s\n",sfile,dfile	Now print the message
system(cpcmd)	Issue the actual cp command using system() , function.

system() function execute given system command. For e.g. if you want to remove file using rm command of Linux, you can write system as follows

```
system("rm foo")
```

OR

```
dcmd = "rm " $1
```

```
system(dcmd)
```

The output of command is not available to program; but *system()* returns the *exit code (error code)* using which you can determine whether command is successful or not. For e.g. We want to see whether rm command is successful or not, you can write code as follows:

```
$ cat > tryrmsys
{
  dcmd = "rm " $1
  if ( system(dcmd) != 0 )
    printf "rm command not successful\n"
  else
    printf "rm command is successful and %s file is removed \n",
    $1
}
```

Run it as (assume that file foo exist and bar does not exist)

```
$ awk -f tryrmsys
```

```
foo
```

```
rm command is successful and foo file is removed
```

```
bar
```

```
rm command not successful
```

(Press CTRL + D to terminate)

Our Second Example:

As I write visual installation guide, I use to capture lot of images for my work, while capturing images I

saved all images (i.e. file names) in UPPER CASE for e.g.

RH7x01.JPG,RH7x02.JPG,...RH7x138.JPG.

Now I would like to rename all files to lowercase then I tried with following two scripts:

[up2low](#) and [rename.awk](#)

up2low can be explained as follows:

Statements/Command	Explanation
AWK_SCRIPT="rename.awk"	Name of awk scripts that renames file
awkspath=\$HOME/bin/\$AWK_SCRIPT	Where our awk script is installed usually it should be installed under your-home-directory/bin (something like /home/vivek/bin)
ls -l > /tmp/file1.\$\$	List all files in current working directory line by line and send output to /tmp/file1.\$\$ file.
tr "[A-Z]" "[a-z]" < /tmp/file1.\$\$ > /tmp/file2.\$\$	Now convert all Uppercase filename to lowercase and store them to /tmp/file2.\$\$ file.
paste /tmp/file1.\$\$ /tmp/file2.\$\$ > /tmp/tmpdb.\$\$	Now paste both Uppercase filename and lowercase filename to third file called /tmp/tmpdb.\$\$ file
rm -f /tmp/file1.\$\$ rm -f /tmp/file2.\$\$	Remove both file1.\$\$ and file2.\$\$ files
if [-f \$awkspath]; then awk -f \$awkspath /tmp/tmpdb.\$\$ else echo -e "\n\$0: Fatal error - \$awkspath not found" echo -e "\nMake sure \$awkspath is set correctly in \$0 script\n" fi	See if rename.awk script installed, if not installed give error message on screen. If installed call the rename.awk script and give it /tmp/tmpdb.\$\$ path to read all filenames from this file.
rm -f /tmp/tmpdb.\$\$	Remove the temporary file.

rename.awk can be explained as follows:

Statements/Command	Explanation

<pre>isdir1 = "[-d " \$1 "] "</pre>	<p>This expression is quite tricky. Its something as follows: <pre>isdir1 = [-d \$1]</pre> Which means see if directory exists using [expr]. As you know [expr] is used to test whether expr is true or not. So we are testing whether directory exist or not. What does \$1 mean? If you remember, in awk \$1 is the first field.</p>
<pre>isdir2 = "[-d " \$2 "] "</pre>	<p>As above except it test for second field as <pre>isdir2 = [-d \$2]</pre> i.e. Whether second field is directory or not.</p>
<pre>scriptname = "up2low" awkscriptname = "rename.awk"</pre>	<p>Our shell script name (up2low) and awk script name (rename.awk).</p>
<pre>sfile = \$1</pre>	<p>Source file</p>
<pre>dfile = \$2</pre>	<p>Destination file</p>
<pre>if (sfile == scriptname sfile == awkscriptname) next</pre>	<p>Make sure we don't accidentally rename our own scripts, if scripts are in current working directory</p>
<pre>else if((system(isdir1)) == 0 system((isdir2)) == 0) { printf "%s or %s is directory can't rename it to lower case\n",sfile,dfile next # continue with next recored }</pre>	<p>Make sure source or destination are files and not the directory. We check this using [expr] command of bash. From the awk script you can called or invoke (as official we called it) the [expr] if directory do exists it will return true (indicated by zero) and if not it will return nonzero value.</p>
<pre>else if (sfile == dfile) { printf "Skiping, \"%s\" is already in lowercase\n",sfile next }</pre>	<p>If both source and destination file are same, it mean file already in lower case no need to rename it to lower case.</p>

```
else # everything is okay rename it to lowercase
{
  mvcmd = "mv " $1 " " $2
  printf "Renaming %s to %s\n",sfile,dfile
  system(mvcmd)
}
```

Now if source and destination files are not

- Directories
- Name of our scripts
- And File is in UPPER CASE

Then rename it to lowercase by issuing command mv command.

Note that if you don't have files name in UPPER case for testing purpose you can create files name as follows:

```
$ for j in 1 2 3 4 5 6 7 8 9 10; do touch TEMP$j.TXT; done
```

Above sample command creates files as TEMP1.TXT,TEMP2.TXT,....TEMP10.TXT files.

Run it as follows:

\$ up2low

Letters or letters is directory can't rename it to lower case

RH6_FILES or rh6_files is directory can't rename it to lower case

Renaming RH7x01.JPG to rh7x01.jpg

Renaming RH7x02.JPG to rh7x02.jpg

Renaming RH7x03.JPG to rh7x03.jpg

Renaming RH7x04.JPG to rh7x04.jpg

Renaming RH7x05.JPG to rh7x05.jpg

Renaming RH7x06.JPG to rh7x06.jpg

....

..

....

Renaming RH7x138.JPG to rh7x138.jpg

On my workstation above output is shown.

[Prev](#)

Loops in awk

[Home](#)

[Up](#)

[Next](#)

awk miscellaneous

awk miscellaneous

You can even take input from keyboard while running awk script, try the following awk script:

```
$ cat > testusrip
BEGIN {
    printf "Your name please:"
    getline na < "-"
    printf "%s your age please:",na
    getline age < "-"
    print "Hello " na, ", next year you will be " age + 1
}
```

Save it and run as

```
$ awk -f testusrip
```

Your name please: Vivek

Vivek your age please: 26

Hello Vivek, next year you will be 27

Here getline function is used to read input from keyboard and then assign the data (inputted from keyboard) to variable.

Syntax:

```
getline variable-name < "-"
```

1	2	3

1 --> getline is function name

2 --> variable-name is used to assign the value read from input

3 --> Means read from stdin (keyboard)

To reading Input from file use following

Syntax:

```
getline < "file-name"
```

Example:

```
getline < "friends.db"
```

To reading Input from pipe use following

Syntax:

```
"command" | getline
```

Example:


```
$ cat > awkread\_file
BEGIN {
    "date" | getline
    print $0
}
```

Run it as

```
$ awk -f awkread_file
```

```
Fri Apr 12 00:05:45 IST 2002
```

Command date is executed and its piped to getline which assign the date command output to variable \$0. If you want your own variable then replace the above program as follows

```
$ cat > awkread\_file1
BEGIN {
    "date" | getline today
    print today
}
```

Run it as follows:

```
$ awk -f awkread_file1
```

Try to understand the following awk script and note down its output.

[temp2final1.awk](#)

[Prev](#)

Real life examples in awk

[Home](#)

[Up](#)

[Next](#)

sed - Quick Introduction

sed - Quick Introduction

SED is a stream editor. A stream editor is used to perform basic text transformations on an input stream (a file or input from a pipeline). SED works by making only one pass over the input(s), and is consequently more efficient. But it is SED's ability to filter text in a pipeline which particularly distinguishes it from other types of editors.

Before getting started with tutorial you must know basic expression which is covered in our [Learning expressions with ex](#) tutorial. For this part of tutorial create [demofile1](#). After creating the file type following sed command at shell prompt:

```
$ sed 's/Linux/UNIX(system v)/' demofile1
```

```
Hello World.
```

```
This is vivek from Poona.
```

```
I love linux.
```

```
.....
```

```
...
```

```
.....
```

```
linux is linux
```

Above sed command can be explained as follows:

Commands	Meaning
sed	Start the sed command
's/Linux/UNIX(system v)/'	Use substitute command to replace Linux with UNIX(system v). General syntax of substitute is s/pattern/pattern-to-substitute/'
demofile1	Read the data from demofile1

General Syntax of sed

Syntax:

```
sed -option 'general expression' [data-file]
```

```
sed -option sed-script-file [data-file]
```

Option can be:

Option	Meaning	Example
-e	Read the different sed command from command line.	\$ sed -e 'sed-commands' data-file-name \$ sed -e 's/Linux/UNIX(system v)/' demofile1
-f	Read the sed command from sed script file.	\$sed -f sed-script-file data-file-name \$ sed -f chgdb.sed friends.tdb

-n	Suppress the output of sed command. When -n is used you must use p command of print flag.	\$ sed -n '/^*..\$/p' demofile2
----	--	---

[Prev](#)

awk miscellaneous

[Home](#)[Up](#)[Next](#)

Redirecting the output of sed command

Redirecting the output of sed command

You can redirect the output of sed command to file as follows

```
$ sed 's/Linux/UNIX(system v)/' demofile1 > file.out
```

And can see the output using cat command as follows

```
$ cat file.out
```

Deleting blank lines

Using sed you can delete all blank line from file as follow

```
$ sed '/^$/d' demofile1
```

As you know pattern `/^$/`, match blank line and **d**, command deletes the blank line.

Following sed command takes input from who command and sed is used to check whether particular user is logged or not.

```
$ who | sed -n '/vivek/p'
```

Here `-n` option to sed command, suppress the output of sed command; and `/vivek/` is the pattern that we are looking for, finally if the pattern found its printed using **p** command of sed.

How to write sed scripts?

Sed command can be grouped together in one text file, this is know as *sed script*. For next example of sed script create [inven1](#) data file and create "*chg1.sed*", script file as follows

Tip: Give *.sed* extension to sed script, *.sh* to Shell script and *.awk* to awk script file(s), this will help you to identify files quickly.

```
$ cat > chg1.sed
1i\
Price of all items changes from 1st-April-2001
/Pen/s/20.00/19.5/
/Pencil/s/2.00/2.60/
/Rubber/s/3.50/4.25/
/Cock/s/45.50/51.00/
```

Run the above sed script as follows:

```
$ sed -f chg1.sed inven1
```

```
Price of all items changes from 1st-April-2001
```

1. Pen 5 19.5
2. Pencil 10 2.60
3. Rubber 3 4.25
4. Cock 2 51.00

In above sed script, the *1i* is the **(i)** insert command. General Syntax is as follows:

Syntax:

```
[line-address]i\
text
```

So,

```
1i\
Price of all items changes from 1st-April-2001
```

means insert the text "Price of all items changes from 1st-April-2001" at line number 1.

Same way you can use append **(a)** or change **(c)** command in your sed script,

General Syntax of append

Syntax:

```
[line-address]a\
text
```

Example:

```
/INDIA/ a\
E-mail: vg@indiamail.co.in
```

Find the word INDIA and append (a) "*E-mail: vg@indiamail.co.in*" text.

General Syntax of change as follows:

Syntax:

```
[line-address]c\  
text
```

Example:

```
/INDIA/ c\  
E-mail: vg@indiamail.co.in
```

Find the word INDIA and change e-mail id to "*vg@indiamail.co.in*"

Rest of the statements (like */Pen/s/20.00/19.5/*) are general substitute statements.

[Prev](#)

Redirecting the output of sed command

[Home](#)

[Up](#)

[Next](#)

More examples of sed

More examples of sed

First create text file [demofile2](#) which is used to demonstrate next sed script examples.

Type following sed command at shell prompt:

```
$ sed -n '/10{2}1/p' demofile2
```

```
1001
```

Above command will print 1001, here in search pattern we have used `{2}`.

Syntax:

```
\{n,\}
```

At least nth occurrences will be matched. So `/10{2}` will look for 1 followed by 0 (zero) and `{2}`, tells sed look for 0 (zero) for twice.

Matcheing any number of occurrence

Syntax:

```
\{n,m\}
```

Matches any number of occurrence between n and m.

Example:

```
$ sed -n '/10{2,4}1/p' demofile2
```

```
1001
```

```
10001
```

```
100001
```

Will match "1001", "10001", "100001" but not "101" or "1000000". Suppose you want to print all line that begins with *** (three stars or asterisks), then you can type command

```
$ sed -n '/^*.$/p' demofile2
```

```
***
```

```
***
```

Above sed expression can be explained as follows:

Command	Explanation
<code>^</code>	Beginning of line
<code> *</code>	Find the asterisk or star (<code>\</code> remove the special meaning of '*' metacharacter)
<code>..</code>	Followed by any two character (you can also use <code>**</code> i.e. <code>\$ sed -n '/^***\$/' demofile2</code>)
<code>\$</code>	End of line (So that only three star or asterisk will be matched)
<code>/p</code>	Print the pattern.

Even you can use following expression for the same purpose

```
$ sed -n '/^*\{2,3\}$/' demofile2
```

Now following command will find out lines between *** and *** and then delete all those line

```
$ sed -e '/^*\{2,3\}$/,/^*\{2,3\}$/' demofile2 > /tmp/fi.$$
```

```
$ cat /tmp/fi.$$
```

Above expression can be explained as follows

Expression	Meaning
^	Beginning of line
*	Find the asterisk or star (\ remove the special meaning of '*' metacharacter)
\{2,3\}	Find next two asterisk
\$	End of line
,	Next range or search pattern
^*\{2,3\}\$	Same as above
d	Now delete all lines between *** and *** range

You can group the commands in sed - scripts as shown following example

```
$ cat > dem_gsed
/^*\{2,3\}$/,/^*\{2,3\}$/{
/^$/d
s/Linux/Linux-Unix/
}<
```

Now save above sed script and run it as follows:

```
$ sed -f dem_gsed demofile2 > /tmp/fi.$$
$ cat /tmp/fi.$$
```

Above sed scripts finds all line between *** and *** and performance following operations

- 1) Delete blank line, if any using /^\$/d expression.
- 2) Substitute "Linux-Unix" for "Linux" word using s/Linux/Linux-Unix/ expression.

Our next example removes all blank line and converts multiple spaces into single space, for this purpose you need [demofile3](#) file. Write sed script as follows:

```
$ cat > rmbksp
/^$/d
s/ */ /g<
```

Run above script as follows:

```
$ sed -f rmbksp demofile3
Welcome to world of sed what sed is?
I don't know what sed is but I think
Rani knows what sed Is
-----
```

Above script can be explained as follows:

Expression	Meaning
/^\$/d	Find all blank line and delete is using d command.
s/ _ */ _g	Find two or more than two blank space and replace it with single blank space

Note that indicates `__` two blank space and indicate `_` one blank space.

For our next and last example create database file [friends](#)

Our task is as follows for friends database file:

- 1) Find all occurrence of "A'bad" word replace it with "Aurangabad" word
- 2) Expand MH state value to Maharashtra
- 3) Find all blank line and replace with actual line (i.e. =====)
- 4) Insert e-mail address of each persons at the end of persons postal address. For each person e-mail ID is different

To achieve all above task write sed script as follows:

```
$ cat > mkchgfrddb
s/A.bad/Aurangabad/g
s/MH/Maharashtra/g
s/^$/===== /g
/V.K. /{
N
N
a\
email:vk@fackmail.co.in
}

/M.M. /{
N
N
a\
email:mm@fackmail.co.in
}

/R.K. /{
N
N
a\
email:rk@fackmail.co.in
}

/A.G. / {
N
N
a\
email:ag@fackmail.co.in
}

/N.K. / {
N
N
a\
email:nk@fackmail.co.in
```

}

Run it as follows:

```
$ sed -f mkchgrddb friends > updated_friendsdb
$ cat updated_friendsdb
```

Above script can be explained as follows:

Expression	Meaning
<code>s/A.bad/Aurangabad/g</code>	Substitute Aurangabad for A'bad. Note that here second character in A'bad is ' (single quote), to match this single quote we have to use . (DOT - Special Metacharcter) that matches any single character.
<code>s/MH/Maharastra/g</code>	Substitute Maharastra for MH
<code>s/^\$/=====/g</code>	Substitute blank line with actual line
<code>/V.K. /{ N N a\ email:vk@fackmail.co.in }</code>	Match the pattern and follow the command between { and }, if pattern found. Here we are finding each friends initial name if it matches then we are going to end of his address (by giving N command twice) and appending (a command) friends e-mail address at the end.

Our last examples shows how we can manipulate text data files using sed. Here our tutorial on sed/awk ends but next version (LSST ver 2.0) will cover more real life examples, case studies using all these tools, plus integration with shell scripts etc.

[Prev](#)

How to write sed scripts?

[Home](#)

[Up](#)

[Next](#)

Examples of Shell Scripts

More examples of Shell Script (Exercise for You :-)

These exercises are to test your general understanding of the shell scripting. My advise is first try to write this shell script yourself so that you understand how to put the concepts to work in real life scripts. For sample answer to exercise you can refer the shell script file supplied with this tutorial. If you want to become the good programmer then your first habit must be to see the good code/samples of programming language then practice lot and finally implement your own code (and become the good programmer!!!).

Q.1. How to write shell script that will add two nos, which are supplied as command line argument, and if this two nos are not given show error and its usage

Answer: [See Q1 shell Script.](#)

Q.2. Write Script to find out biggest number from given three nos. Nos are supplied as command line argument. Print error if sufficient arguments are not supplied.

Answer: [See Q2 shell Script.](#)

Q.3. Write script to print nos as 5,4,3,2,1 using while loop.

Answer: [See Q3 shell Script.](#)

Q.4. Write Script, using case statement to perform basic math operation as follows

+ addition
- subtraction
x multiplication
/ division

The name of script must be 'q4' which works as follows

\$./q4 20 / 3, Also check for sufficient command line arguments

Answer: [See Q4 shell Script.](#)

Q.5. Write Script to see current date, time, username, and current directory

Answer: [See Q5 shell Script.](#)

Q.6. Write script to print given number in reverse order, for eg. If no is 123 it must print as 321.

Answer: [See Q6 shell Script.](#)

Q.7. Write script to print given numbers sum of all digit, For eg. If no is 123 it's sum of all digit will be $1+2+3 = 6$.

Answer: [See Q7 shell Script.](#)

Q.8. How to perform real number (number with decimal point) calculation in Linux

Answer: Use Linux's bc command

Q.9. How to calculate $5.12 + 2.5$ real number calculation at \$ prompt in Shell ?

Answer: Use command as , \$ echo 5.12 + 2.5 | bc , here we are giving echo commands output to bc to calculate the $5.12 + 2.5$

Q.10. How to perform real number calculation in shell script and store result to third variable , lets say $a=5.66$, $b=8.67$, $c=a+b$?

Answer: [See Q10 shell Script.](#)

Q.11. Write script to determine whether given file exist or not, file name is supplied as command line argument, also check for sufficient number of command line argument

Answer: [See Q11 shell Script.](#)

Q.12. Write script to determine whether given command line argument (\$1) contains "*" symbol or not, if \$1 does not contains "*" symbol add it to \$1, otherwise show message "Symbol is not required". For e.g. If we called this script Q12 then after giving ,

\$ Q12 /bin
Here \$1 is /bin, it should check whether "*" symbol is present or not if not it should print Required i.e. /bin/*, and if symbol present then Symbol is not required must be printed. Test your script as
\$ Q12 /bin

\$ Q12 /bin/*

Answer: [See Q12 shell Script](#)

Q.13. Write script to print contains of file from given line number to next given number of lines. For e.g. If we called this script as Q13 and run as

\$ Q13 5 5 myf , Here print contains of 'myf' file from line number 5 to next 5 line of that file.

Answer: [See Q13 shell Script](#)

Q.14. Write script to implement getopts statement, your script should understand following command line argument called this script Q14,

Q14 -c -d -m -e

Where options work as

-c clear the screen

-d show list of files in current working directory

-m start mc (midnight commander shell) , if installed

-e { editor } start this { editor } if installed

Answer: [See Q14 shell Script](#)

Q.15. Write script called sayHello, put this script into your startup file called .bash_profile, the script should run as soon as you logon to system, and it print any one of the following message in infobox using dialog utility, if installed in your system, If dialog utility is not installed then use echo statement to print message : -

Good Morning

Good Afternoon

Good Evening , according to system time.

Answer: [See Q15 shell Script](#)

Q.16. How to write script, that will print, Message "Hello World" , in Bold and Blink effect, and in different colors like red, brown etc using echo command.

Answer: [See Q16 shell Script](#)

Q.17. Write script to implement background process that will continually print current time in upper right corner of the screen , while user can do his/her normal job at \$ prompt.

Answer: [See Q17 shell Script.](#)

Q.18. Write shell script to implement menus using dialog utility. Menu-items and action according to select menu-item is as follows:

Menu-Item	Purpose	Action for Menu-Item
Date/time	To see current date time	Date and time must be shown using infobox of dialog utility
Calendar	To see current calendar	Calendar must be shown using infobox of dialog utility
Delete	To delete selected file	First ask user name of directory where all files are present, if no name of directory given assumes current directory, then show all files only of that directory, Files must be shown on screen using menus of dialog utility, let the user select the file, then ask the confirmation to user whether he/she wants to delete selected file, if answer is yes then delete the file , report errors if any while deleting file to user.
Exit	To Exit this shell script	Exit/Stops the menu driven program i.e. this script

Note: Create function for all action for e.g. To show date/time on screen create function show_datetime().

Answer: [See Q18 shell Script.](#)

Q.19. Write shell script to show various system configuration like

- 1) Currently logged user and his logname
- 2) Your current shell
- 3) Your home directory
- 4) Your operating system type
- 5) Your current path setting
- 6) Your current working directory
- 7) Show Currently logged number of users
- 8) About your os and version ,release number , kernel version
- 9) Show all available shells

Introduction

This is new chapter added to LSST v1.05r3, its gives more references to other material available on shell scripting on Net or else ware. It also indicates some other resources which might be useful while programming the shell.

Appendix - A	Information
Appendix - A Linux File Server Tutorial (LFST) version b0.1 Rev. 2	This tutorial/document is useful for beginners who wish to learn Linux file system, it covers basic concept of file system, commands or utilities related with file system. It will explain basic file concepts such as what is file & directories, what are the mount points, how to use cdrom or floppy drive under Linux.
Appendix - B Linux Command Reference (LCR)	This command reference is specially written for the LSST. It contains command name, general syntax followed by an example. This is useful while programming shell and you can use as Quick Linux Command Reference guide.

More information on upcoming edition of this tutorial.

About Author

Vivek G. Gite runs small firm called "[Cyberciti Computers](#)" and *nix Solution firm [nixCraft](#). He is freelance software developer and also teaches computer hardware, networking and Linux/Unix to beginners. He is also working with various Computers Firms as Technology Consultant. Currently he writes article on Linux/Unix, LSST is one of such article/document. His future plan includes more article/documents on Linux especially for beginners. If you have any suggestion or new ideas or problem with this tutorial, please feel free to contact author using following e-mail ID.

How do I contact the author?

I can be contacted by e-mail: vivek@nixcraft.com.

Where do I find the latest version?

Please visit <http://www.cyberciti.biz/nixcraft/linux/docs/> for latest version of this Tutorial/Document as well as for other tutorial/documents.

Other Information

This tutorial is prepared with help of all valuable material from web as well as from on-line help of Linux (man and info pages), Linux how-to's etc. Also special thanks to Ashish for his valuable suggestion for this tutorial/document.

All the trademarks are acknowledged and used for identification purpose only.

[Prev](#)

About this Document

This document is Copyright (C) 1999, 2000, 2001,2002 by Vivek G. Gite <vivek@nixcraft.com>. It may be freely distributed in any medium as long as the text (including this notice) is kept intact and the content is not modified, edited, added to or otherwise changed. Formatting and presenting may be modified. Small excerpts may be made as long as the full document is properly and conspicuously referenced.

If you do the mirror of this document, please send e-mail to the address above, so that you can be informed of updates.

All trademark within are property of their respective holders.

Although the author believes the contents to be accurate at the time of publication, no liability is assumed for them, their application or any consequences thereof. if any misrepresentations, errors or other need of clarification is found, please contact the author immediately.

The latest copy of this document can always be obtained from:

<http://www.cyberciti.biz/nixcraft/linux/docs/>

Last updated [Linux Shell Scripting Tutorial v1.05r3 \(LSST\)](#) - on Thu., July, 04, 2002.

[Prev](#)

About the author

[Home](#)[Up](#)

An UniqLinux Features


```
#!/bin/bash
#
# Linux Shell Scripting Tutorial 1.05r3, Summer-2002
#
# Written by Vivek G. Gite <vivek@nixcraft.com>
#
# Latest version can be found at http://www.nixcraft.com/
#
# Q1.Script to sum to nos
#

if [ $# -ne 2 ]
then
    echo "Usage - $0 x y"
    echo "          Where x and y are two nos for which I will print sum"
    exit 1
fi
    echo "Sum of $1 and $2 is `expr $1 + $2`"
#
# ./ch.sh: vivek-tech.com to nixcraft.com referance converted using this tool
# See the tool at http://www.nixcraft.com/uniqlinuxfeatures/tools/
#
```



```
#!/bin/bash
#
# Linux Shell Scripting Tutorial 1.05r3, Summer-2002
#
# Written by Vivek G. Gite <vivek@nixcraft.com>
#
# Latest version can be found at http://www.nixcraft.com/
#
# Q3
# Algo:
#     1) START: set value of i to 5 (since we want to start from 5, if you
#         want to start from other value put that value)
#     2) Start While Loop
#     3) Check, Is value of i is zero, If yes goto step 5 else
#         continue with next step
#     4) print i, decement i by 1 (i.e. i=i-1 to goto zero) and
#         goto step 3
#     5) END
#
i=5
while test $i != 0
do
    echo "$i"
    "
    i=`expr $i - 1`
done
#
# ./ch.sh: vivek-tech.com to nixcraft.com referance converted using this tool
# See the tool at http://www.nixcraft.com/uniqlinuxfeatures/tools/
#
```

```
#!/bin/bash
#
# Linux Shell Scripting Tutorial 1.05r3, Summer-2002
#
# Written by Vivek G. Gite <vivek@nixcraft.com>
#
# Latest version can be found at http://www.nixcraft.com/
#
# Q4
#

if test $# = 3
then
    case $2 in
        +) let z=$(( $1+$3 ));
        -) let z=$(( $1-$3 ));
        /) let z=$(( $1/$3 ));
        x|X) let z=$(( $1*$3 ));
        *) echo Warning - $2 invalied operator, only +,-,x,/ operator allowed
           exit;;
    esac
    echo Answer is $z
else
    echo "Usage - $0 value1 operator value2"
    echo "          Where, value1 and value2 are numeric values"
    echo "          operator can be +,-,/,x (For Multiplication)"
fi

#
# ./ch.sh: vivek-tech.com to nixcraft.com referance converted using this tool
# See the tool at http://www.nixcraft.com/uniqlinuxfeatures/tools/
#
```

```
#!/bin/bash
#
# Linux Shell Scripting Tutorial 1.05r3, Summer-2002
#
# Written by Vivek G. Gite <vivek@nixcraft.com>
#
# Latest version can be found at http://www.nixcraft.com/
#
# Q5
#
echo "Hello, $LOGNAME"
echo "Current date is `date`"
echo "User is `who i am`"
echo "Current direcotry `pwd`"

#
# ./ch.sh: vivek-tech.com to nixcraft.com referance converted using this tool
# See the tool at http://www.nixcraft.com/uniqlinuxfeatures/tools/
#
```

```
#!/bin/bash
#
# Linux Shell Scripting Tutorial 1.05r3, Summer-2002
#
# Written by Vivek G. Gite <vivek@nixcraft.com>
#
# Latest version can be found at http://www.nixcraft.com/
#
# Script to reverse given no
#
# Algo:
#     1) Input number n
#     2) Set rev=0, sd=0
#     3) Find single digit in sd as n % 10 it will give (left most digit)
#     4) Construct revrse no as rev * 10 + sd
#     5) Decrment n by 1
#     6) Is n is greater than zero, if yes goto step 3, otherwise next step
#     7) Print rev
#
if [ $# -ne 1 ]
then
    echo "Usage: $0    number"
    echo "    I will find reverse of given number"
    echo "    For eg. $0 123, I will print 321"
    exit 1
fi

n=$1
rev=0
sd=0

while [ $n -gt 0 ]
do
    sd=`expr $n % 10`
    rev=`expr $rev \* 10 + $sd`
    n=`expr $n / 10`
done
    echo "Reverse number is $rev"

#
# ./ch.sh: vivek-tech.com to nixcraft.com referance converted using this tool
# See the tool at http://www.nixcraft.com/uniqlinuxfeatures/tools/
#
```

```
#!/bin/bash
#
# Linux Shell Scripting Tutorial 1.05r3, Summer-2002
#
# Written by Vivek G. Gite <vivek@nixcraft.com>
#
# Latest version can be found at http://www.nixcraft.com/
#
# Algo:
#     1) Input number n
#     2) Set sum=0, sd=0
#     3) Find single digit in sd as n % 10 it will give (left most digit)
#     4) Construct sum no as sum=sum+sd
#     5) Decrment n by 1
#     6) Is n is greater than zero, if yes goto step 3, otherwise next step
#     7) Print sum
#
if [ $# -ne 1 ]
then
    echo "Usage: $0  number"
    echo "        I will find sum of all digit for given number"
    echo "        For eg. $0 123, I will print 6 as sum of all digit (1+2+3)"
    exit 1
fi

n=$1
sum=0
sd=0
while [ $n -gt 0 ]
do
    sd=`expr $n % 10`
    sum=`expr $sum + $sd`
    n=`expr $n / 10`
done
    echo "Sum of digit for numner is $sum"

#
# ./ch.sh: vivek-tech.com to nixcraft.com referance converted using this tool
# See the tool at http://www.nixcraft.com/uniqlinuxfeatures/tools/
#
```

```
#!/bin/bash
#
# Linux Shell Scripting Tutorial 1.05r3, Summer-2002
#
# Written by Vivek G. Gite <vivek@nixcraft.com>
#
# Latest version can be found at http://www.nixcraft.com/
#
# Q10
#
a=5.66
b=8.67
c=`echo $a + $b | bc`
echo "$a + $b = $c"

#
# ./ch.sh: vivek-tech.com to nixcraft.com referance converted using this tool
# See the tool at http://www.nixcraft.com/uniqlinuxfeatures/tools/
#
```



```
#!/bin/bash
#
# Linux Shell Scripting Tutorial 1.05r3, Summer-2002
#
# Written by Vivek G. Gite <vivek@nixcraft.com>
#
# Latest version can be found at http://www.nixcraft.com/
#
# Q11

if [ $# -ne 1 ]
then
    echo "Usage - $0 file-name"
    exit 1
fi

if [ -f $1 ]
then
    echo "$1 file exist"
else
    echo "Sorry, $1 file does not exist"
fi

#
# ./ch.sh: vivek-tech.com to nixcraft.com referance converted using this tool
# See the tool at http://www.nixcraft.com/uniqlinuxfeatures/tools/
#
```

```
#!/bin/bash
#
# Linux Shell Scripting Tutorial 1.05r3, Summer-2002
#
# Written by Vivek G. Gite <vivek@nixcraft.com>
#
# Latest version can be found at http://www.nixcraft.com/
#
# Q12
# Script to check whether "/" is included, in $1 or not
#

cat "$1" > /tmp/file.$$    2>/tmp/file0.$$

grep "/" /tmp/file.$$    >/tmp/file0.$$

if [ $? -eq 1 ]
then
    echo "Required i.e. $1/"
else
    echo "Symbol is Not required"
fi

rm -f /tmp/file.$$
rm -f /tmp/file0.$$
#
# ./ch.sh: vivek-tech.com to nixcraft.com referance converted using this tool
# See the tool at http://www.nixcraft.com/uniqlinuxfeatures/tools/
#
```

```
#!/bin/bash
#
# Linux Shell Scripting Tutorial 1.05r3, Summer-2002
#
# Written by Vivek G. Gite <vivek@nixcraft.com>
#
# Latest version can be found at http://www.nixcraft.com/
#
# Q13
#
# Shell script to print contents of file from given line no to next
# given numberlines
#
#
# Print error / diagnostic for user if no arg's given
#
if [ $# -eq 0 ]
then
    echo "$0:Error command arguments missing!"
    echo "Usage: $0 start_line upto_line filename"
    echo "Where start_line is line number from which you would like to print file"
    echo "upto_line is line number upto which would like to print"
    echo "For eg. $0 5 5 myfile"
    echo "Here from myfile total 5 lines printed starting from line no. 5 to"
    echo "line no 10."
    exit 1
fi

#
# Look for sufficient arg's
#

if [ $# -eq 3 ]; then
    if [ -e $3 ]; then
        tail +$1 $3 | head -n$2
    else
        echo "$0: Error opening file $3"
        exit 2
    fi
else
    echo "Missing arguments!"
fi

#
# ./ch.sh: vivek-tech.com to nixcraft.com reference converted using this tool
# See the tool at http://www.nixcraft.com/uniqlinuxfeatures/tools/
#
```

```
#!/bin/bash
#
# Linux Shell Scripting Tutorial 1.05r3, Summer-2002
#
# Written by Vivek G. Gite <vivek@nixcraft.com>
#
# Latest version can be found at http://www.nixcraft.com/
#
# Q14
# -c clear
# -d dir
# -m mc
# -e vi { editor }
#
#
# Function to clear the screen
#
cls()
{
    clear
    echo "Clear screen, press a key . . ."
    read
    return
}
#
# Function to show files in current directory
#
show_ls()
{
    ls
    echo "list files, press a key . . ."
    read
    return
}
#
# Function to start mc
#
start_mc()
{
    if which mc > /dev/null ; then
        mc
        echo "Midnight commander, Press a key . . ."
        read
    else
        echo "Error: Midnight commander not installed, Press a key . . ."
        read
    fi
    return
}
#
# Function to start editor
#
start_ed()
{
    ced=$1
    if which $ced > /dev/null ; then
        $ced
    fi
}
```

```
    echo "$ced, Press a key . . ."
    read
else
    echo "Error: $ced is not installed or no such editor exist, Press a key . . ."
    read
fi
return
}

#
# Function to print help
#
print_help_uu()
{
    echo "Usage: $0 -c -d -m -v {editor name}";
    echo "Where -c clear the screen";
    echo "        -d show dir";
    echo "        -m start midnight commander shell";
    echo "        -e {editor}, start {editor} of your choice";
    return
}

#
# Main procedure start here
#
# Check for sufficient args
#

if [ $# -eq 0 ] ; then
    print_help_uu
    exit 1
fi

#
# Now parse command line arguments
#
while getopts cdme: opt
do
    case "$opt" in
        c) cls;;
        d) show_ls;;
        m) start_mc;;
        e) thised="$OPTARG"; start_ed $thised ;;
        \?) print_help_uu; exit 1;;
    esac
done

#
# ./ch.sh: vivek-tech.com to nixcraft.com referance converted using this tool
# See the tool at http://www.nixcraft.com/uniqlinuxfeatures/tools/
#
```

```
#!/bin/bash
#
# Linux Shell Scripting Tutorial 1.05r3, Summer-2002
#
# Written by Vivek G. Gite <vivek@nixcraft.com>
#
# Latest version can be found at http://www.nixcraft.com/
#
# Q15
#

temph=`date | cut -c12-13`
dat=`date +"%A %d in %B of %Y (%r)"`

if [ $temph -lt 12 ]
then
    mess="Good Morning $LOGNAME, Have nice day!"
fi

if [ $temph -gt 12 -a $temph -le 16 ]
then
    mess="Good Afternoon $LOGNAME"
fi

if [ $temph -gt 16 -a $temph -le 18 ]
then
    mess="Good Evening $LOGNAME"
fi

if which dialog > /dev/null
then
    dialog --backtitle "Linux Shell Script Tutorial"\
    --title "(-: Welcome to Linux :-)"\
    --infobox "\n$mess\nThis is $dat" 6 60
    echo -n "                                Press a key to continue. . .
"
    read
    clear
else
    echo -e "$mess\nThis is $dat"
fi

#
# ./ch.sh: vivek-tech.com to nixcraft.com referance converted using this tool
# See the tool at http://www.nixcraft.com/uniqlinuxfeatures/tools/
#
```



```
#!/bin/bash
#
# Linux Shell Scripting Tutorial 1.05r3, Summer-2002
#
# Written by Vivek G. Gite <vivek@nixcraft.com>
#
# Latest version can be found at http://www.nixcraft.com/
#
# Q17
# To run type at $ promot as
# $ q17 &
#

echo
echo "Digital Clock for Linux"
echo "To stop this clock use command kill pid, see above for pid"
echo "Press a key to continue. . ."

while :
do
    ti=`date +"%r" `
    echo -e -n "\033[7s"      #save current screen postion & attributes
    #
    # Show the clock
    #

    tput cup 0 69           # row 0 and column 69 is used to show clock

    echo -n $ti            # put clock on screen

    echo -e -n "\033[8u"    #restore current screen postion & attributs
    #
    #Delay fro 1 second
    #
    sleep 1
done

#
# ./ch.sh: vivek-tech.com to nixcraft.com referance converted using this tool
# See the tool at http://www.nixcraft.com/uniqlinuxfeatures/tools/
#
```



```
#!/bin/bash
#
# Linux Shell Scripting Tutorial 1.05r3, Summer-2002
#
# Written by Vivek G. Gite <vivek@nixcraft.com>
#
# Latest version can be found at http://www.nixcraft.com/
#

show_datetime()
{
    dialog --backtitle "Linux Shell Tutorial" --title "System date and Time" --infobox
"Date is `date`" 3 40
    read
    return
}

show_cal()
{
    cal > menuchoice.temp.$$
    dialog --backtitle "Linux Shell Tutorial" --title "Calender" --infobox "`cat
menuchoice.temp.$$`" 9 25
    read
    rm -f menuchoice.temp.$$
    return
}

delete_file()
{
    dialog --backtitle "Linux Shell Tutorial" --title "Delete file"\
--inputbox "Enter directory path (Enter for Current Directory)"\
10 40 2>/tmp/dirip.$$
    rtval=$?

    case $rtval in
        1) rm -f /tmp/dirip.$$ ; return ;;
        255) rm -f /tmp/dirip.$$ ; return ;;
    esac

    mfile=`cat /tmp/dirip.$$`

    if [ -z $mfile ]
    then
        mfile=`pwd`/*
    else
        grep "*" /tmp/dirip.$$
        if [ $? -eq 1 ]
        then
            mfile=$mfile/*
        fi
    fi

    for i in $mfile
    do
        if [ -f $i ]
        then
            echo "$i Delete?" >> /tmp/finallist.$$
        fi
    done
}
```

```
dialog --backtitle "Linux Shell Tutorial" --title "Select File to Delete"\
--menu "Use [Up][Down] to move, [Enter] to select file"\
20 60 12 `cat /tmp/finallist.$$` 2>/tmp/file2delete.tmp.$$
```

```
rtval=$?
```

```
file2erase=`cat /tmp/file2delete.tmp.$$`
```

```
case $rtval in
```

```
0) dialog --backtitle "Linux Shell Tutorial" --title "Are you shur"\
--yesno "\n\nDo you want to delete : $file2erase " 10 60
```

```
if [ $? -eq 0 ] ; then
```

```
rm -f $file2erase
```

```
if [ $? -eq 0 ] ; then
```

```
dialog --backtitle "Linux Shell Tutorial"\
```

```
--title "Information: Delete Command" --infobox "File: $file2erase is
```

```
Successfully deleted,Press a key" 5 60
```

```
read
```

```
else
```

```
dialog --backtitle "Linux Shell Tutorial"\
```

```
--title "Error: Delete Command" --infobox "Error deleting File: $file2erase,
```

```
Press a key" 5 60
```

```
read
```

```
fi
```

```
else
```

```
dialog --backtitle "Linux Shell Tutorial"\
```

```
--title "Information: Delete Command" --infobox "File: $file2erase is not
```

```
deleted, Action is canceled, Press a key" 5 60
```

```
read
```

```
fi
```

```
;;
```

```
1) rm -f /tmp/dirip.$$ ; rm -f /tmp/finallist.$$ ;
```

```
rm -f /tmp/file2delete.tmp.$$; return;;
```

```
255) rm -f /tmp/dirip.$$ ; rm -f /tmp/finallist.$$ ;
```

```
rm -f /tmp/file2delete.tmp.$$; return;;
```

```
esac
```

```
rm -f /tmp/dirip.$$
```

```
rm -f /tmp/finallist.$$
```

```
rm -f /tmp/file2delete.tmp.$$
```

```
return
```

```
}
```

```
while true
```

```
do
```

```
dialog --clear --title "Main Menu" \
```

```
--menu "To move [UP/DOWN] arrow keys \n\
```

```
[Enter] to Select\n\
```

```
Choose the Service you like:" 20 51 4 \
```

```
"Date/time" "To see System Date & Time" \
```

```
"Calender" "To see Calaender"\
```

```
"Delete" "To remove file"\
```

```
"Exit" "To exit this Program" 2> menuchoice.temp.$$
```

```
retopt=$?
```

```
choice=`cat menuchoice.temp.$$`
```

```
rm -f menuchoice.temp.$$
```

```
case $reopt in
  0)
    case $choice in
      Date/time) show_datetime ;;
      Calender) show_cal ;;
      Delete) delete_file ;;
      Exit) exit 0;;
    esac
    ;;
  1) exit ;;
  255) exit ;;
esac
done
clear

#
# ./ch.sh: vivek-tech.com to nixcraft.com referance converted using this tool
# See the tool at http://www.nixcraft.com/uniqlinuxfeatures/tools/
#
```

```
#!/bin/bash
#
# Linux Shell Scripting Tutorial 1.05r3, Summer-2002
#
# Written by Vivek G. Gite <vivek@nixcraft.com>
#
# Latest version can be found at http://www.nixcraft.com/
#
# Q19
#

nouser=`who | wc -l`
echo -e "User name: $USER (Login name: $LOGNAME)" >> /tmp/info.tmp.01.$$$
echo -e "Current Shell: $SHELL" >> /tmp/info.tmp.01.$$$
echo -e "Home Directory: $HOME" >> /tmp/info.tmp.01.$$$
echo -e "Your O/s Type: $OSTYPE" >> /tmp/info.tmp.01.$$$
echo -e "PATH: $PATH" >> /tmp/info.tmp.01.$$$
echo -e "Current directory: `pwd`" >> /tmp/info.tmp.01.$$$
echo -e "Currently Logged: $nouser user(s)" >> /tmp/info.tmp.01.$$$

if [ -f /etc/redhat-release ]
then
    echo -e "OS: `cat /etc/redhat-release`" >> /tmp/info.tmp.01.$$$
fi

if [ -f /etc/shells ]
then
    echo -e "Available Shells: " >> /tmp/info.tmp.01.$$$
    echo -e "`cat /etc/shells`" >> /tmp/info.tmp.01.$$$
fi

if [ -f /etc/sysconfig/mouse ]
then
    echo -e "-----" >>
/tmp/info.tmp.01.$$$
    echo -e "Computer Mouse Information: " >> /tmp/info.tmp.01.$$$
    echo -e "-----" >>
/tmp/info.tmp.01.$$$
    echo -e "`cat /etc/sysconfig/mouse`" >> /tmp/info.tmp.01.$$$
fi
echo -e "-----" >>
/tmp/info.tmp.01.$$$
echo -e "Computer CPU Information:" >> /tmp/info.tmp.01.$$$
echo -e "-----" >>
/tmp/info.tmp.01.$$$
cat /proc/cpuinfo >> /tmp/info.tmp.01.$$$

echo -e "-----" >>
/tmp/info.tmp.01.$$$
echo -e "Computer Memory Information:" >> /tmp/info.tmp.01.$$$
echo -e "-----" >>
/tmp/info.tmp.01.$$$
cat /proc/meminfo >> /tmp/info.tmp.01.$$$

if [ -d /proc/ide/hda ]
then
    echo -e "-----" >>
/tmp/info.tmp.01.$$$
    echo -e "Hard disk information:" >> /tmp/info.tmp.01.$$$
    echo -e "-----" >>
/tmp/info.tmp.01.$$$
```

```
echo -e "Model: `cat /proc/ide/hda/model` " >> /tmp/info.tmp.01.$$$  
echo -e "Driver: `cat /proc/ide/hda/driver` " >> /tmp/info.tmp.01.$$$  
echo -e "Cache size: `cat /proc/ide/hda/cache` " >> /tmp/info.tmp.01.$$$
```

```
fi
```

```
echo -e "-----" >>
```

```
/tmp/info.tmp.01.$$$
```

```
echo -e "File System (Mount):" >> /tmp/info.tmp.01.$$$
```

```
echo -e "-----" >>
```

```
/tmp/info.tmp.01.$$$
```

```
cat /proc/mounts >> /tmp/info.tmp.01.$$$
```

```
if which dialog > /dev/null
```

```
then
```

```
    dialog --backtitle "Linux Software Diagnostics (LSD) Shell Script Ver.1.0" --title  
"Press Up/Down Keys to move" --textbox /tmp/info.tmp.01.$$$ 21 70
```

```
else
```

```
    cat /tmp/info.tmp.01.$$$ |more
```

```
fi
```

```
rm -f /tmp/info.tmp.01.$$$
```

```
#
```

```
# ./ch.sh: vivek-tech.com to nixcraft.com referance converted using this tool
```

```
# See the tool at http://www.nixcraft.com/uniqlinuxfeatures/tools/
```

```
#
```

```
#!/bin/bash
#
# Linux Shell Scripting Tutorial 1.05r3, Summer-2002
#
# Written by Vivek G. Gite <vivek@nixcraft.com>
#
# Latest version can be found at http://www.nixcraft.com/
#

echo "Can you see the following:"

for (( i=1; i<=5; i++ ))
do
    for (( j=1; j<=i; j++ ))
    do
        echo -n "$i"
    done
    echo " "
done

#
# ./ch.sh: vivek-tech.com to nixcraft.com referance converted using this tool
# See the tool at http://www.nixcraft.com/uniqlinuxfeatures/tools/
#
```

```
#!/bin/bash
#
# Linux Shell Scripting Tutorial 1.05r3, Summer-2002
#
# Written by Vivek G. Gite <vivek@nixcraft.com>
#
# Latest version can be found at http://www.nixcraft.com/
#

echo "Can you see the following:"

for (( i=1; i<=5; i++ ))
do
    for (( j=1; j<=i; j++ ))
    do
        echo -n "$j"
    done
    echo " "
done

#
# ./ch.sh: vivek-tech.com to nixcraft.com referance converted using this tool
# See the tool at http://www.nixcraft.com/uniqlinuxfeatures/tools/
#
```

```
#!/bin/bash
#
# Linux Shell Scripting Tutorial 1.05r3, Summer-2002
#
# Written by Vivek G. Gite <vivek@nixcraft.com>
#
# Latest version can be found at http://www.nixcraft.com/
#

echo "Climb the steps of success"

for (( i=1; i<=5; i++ ))
do
    for (( j=1; j<=i; j++ ))
    do
        echo -n " |"
    done
    echo "_ "
done

#
# ./ch.sh: vivek-tech.com to nixcraft.com referance converted using this tool
# See the tool at http://www.nixcraft.com/uniqlinuxfeatures/tools/
#
```



```
#!/bin/bash
#
# Linux Shell Scripting Tutorial 1.05r3, Summer-2002
#
# Written by Vivek G. Gite <vivek@nixcraft.com>
#
# Latest version can be found at http://www.nixcraft.com/
#

echo "Stars"

for (( i=1; i<=5; i++ ))
do
    for (( j=1; j<=i; j++ ))
    do
        echo -n " *"
    done
    echo ""
done

#
# ./ch.sh: vivek-tech.com to nixcraft.com referance converted using this tool
# See the tool at http://www.nixcraft.com/uniqlinuxfeatures/tools/
#
```

```
#!/bin/bash
#
# Linux Shell Scripting Tutorial 1.05r3, Summer-2002
#
# Written by Vivek G. Gite <vivek@nixcraft.com>
#
# Latest version can be found at http://www.nixcraft.com/
#

echo "Stars"

for (( i=1; i<=5; i++ ))
do
    for (( j=1; j<=i; j++ ))
    do
        echo -n " *"
    done
    echo ""
done

for (( i=5; i>=1; i-- ))
do
    for (( j=1; j<=i; j++ ))
    do
        echo -n " *"
    done
    echo ""
done

#
# ./ch.sh: vivek-tech.com to nixcraft.com referance converted using this tool
# See the tool at http://www.nixcraft.com/uniqlinuxfeatures/tools/
#
```

```
#!/bin/bash
#
# Linux Shell Scripting Tutorial 1.05r3, Summer-2002
#
# Written by Vivek G. Gite <vivek@nixcraft.com>
#
# Latest version can be found at http://www.nixcraft.com/
#

clear

for (( i=1; i<=3; i++ ))
do
    for (( j=1; j<=i; j++ ))
    do
        echo -n "|Linux"
    done
    echo "_____"
done

for (( i=3; i>=1; i-- ))
do
    for (( j=1; j<=i; j++ ))
    do
        echo -n "|Linux"
    done

    if [ $i -eq 3 ]; then
        echo -n "_____"
        echo -n -e ">> Powerd Server.\n"
    else
        echo "~~~~~"
    fi
done

#
# ./ch.sh: vivek-tech.com to nixcraft.com referance converted using this tool
# See the tool at http://www.nixcraft.com/uniqlinuxfeatures/tools/
#
```

```
#!/bin/bash
#
# Linux Shell Scripting Tutorial 1.05r3, Summer-2002
#
# Written by Vivek G. Gite <vivek@nixcraft.com>
#
# Latest version can be found at http://www.nixcraft.com/
#

MAX_NO=0

echo -n "Enter Number between (5 to 9) : "
read MAX_NO

if ! [ $MAX_NO -ge 5 -a $MAX_NO -le 9 ] ; then
    echo "I ask to enter number between 5 and 9, Okay"
    exit 1
fi

clear

for (( i=1; i<=MAX_NO; i++ ))
do
    for (( s=MAX_NO; s>=i; s-- ))
    do
        echo -n " "
    done
    for (( j=1; j<=i; j++ ))
    do
        echo -n " $i"
    done
    echo ""
done

for (( i=1; i<=MAX_NO; i++ ))
do
    for (( s=MAX_NO; s>=i; s-- ))
    do
        echo -n " "
    done
    for (( j=1; j<=i; j++ ))
    do
        echo -n " ."
    done
    echo ""
done

echo -e "\n\n\t\t\tI hope you like it my stupidity (?)"

#
# ./ch.sh: vivek-tech.com to nixcraft.com referance converted using this tool
# See the tool at http://www.nixcraft.com/uniqlinuxfeatures/tools/
#
```

```
#!/bin/bash
#
# Linux Shell Scripting Tutorial 1.05r3, Summer-2002
#
# Written by Vivek G. Gite <vivek@nixcraft.com>
#
# Latest version can be found at http://www.nixcraft.com/
#

MAX_NO=0

echo -n "Enter Number between (5 to 9) : "
read MAX_NO

if ! [ $MAX_NO -ge 5 -a $MAX_NO -le 9 ] ; then
    echo "I ask to enter number between 5 and 9, Okay"
    exit 1
fi

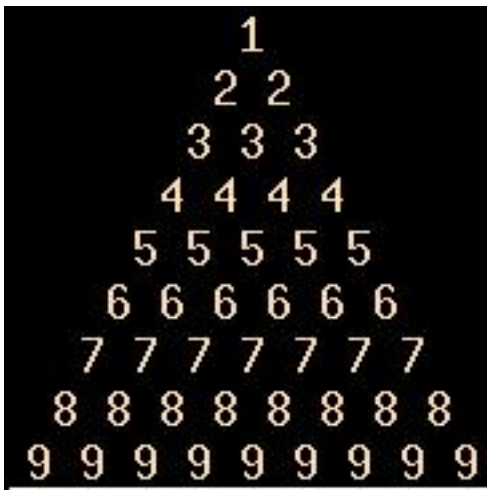
clear

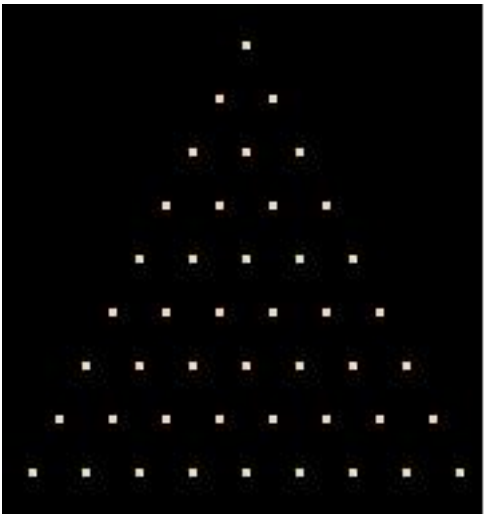
for (( i=1; i<=MAX_NO; i++ ))
do
    for (( s=MAX_NO; s>=i; s-- ))
    do
        echo -n " "
    done
    for (( j=1; j<=i; j++ ))
    do
        echo -n " ."
    done
    echo ""
done
##### Second stage #####
##
##
for (( i=MAX_NO; i>=1; i-- ))
do
    for (( s=i; s<=MAX_NO; s++ ))
    do
        echo -n " "
    done
    for (( j=1; j<=i; j++ ))
    do
        echo -n " ."
    done
    echo ""
done

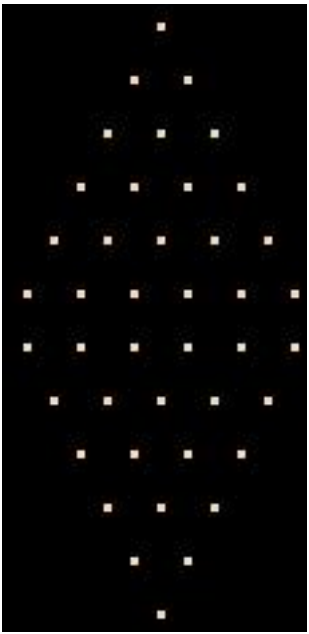
echo -e "\n\n\t\t\tI hope you like it my stupidity (?)"

#
# ./ch.sh: vivek-tech.com to nixcraft.com referance converted using this tool
# See the tool at http://www.nixcraft.com/uniqlinuxfeatures/tools/
#
```

```
|Linux_____  
|Linux|Linux_____  
|Linux|Linux|Linux_____  
|Linux|Linux|Linux_____>> Powerd Server.  
|Linux|Linux~~~~~  
|Linux~~~~~
```







```
# Linux Shell Scripting Tutorial v1.05r3, Summer-2002
# rename.awk : awk script to rename file with some builtin Intelligence
# Author : Vivek G. Gite <vivek@nixcraft.com>
#
BEGIN{
}

#
# main logic is here
#
{
    isdir1 = "[ -d " $1 " ] "
    isdir2 = "[ -d " $2 " ] "

    scriptname = "up2low"
    awkscriptname = "rename.awk"

    sfile = $1
    dfile = $2
    #
    # we are not suppose to rename dirs in source or destination
    #

    #
    # make sure we are renaming our self if in same dir
    #
    if ( sfile == scriptname || sfile == awkscriptname )
        next
    else if( ( system(isdir1) ) == 0 || system((isdir2)) == 0 )
    {
        printf "%s or %s is directory can't rename it to lower case\n",sfile,dfile
        next # continue with next recored
    }
    else if ( sfile == dfile )
    {
        printf "Skiping, \"%s\" is already in lowercase\n",sfile
        next
    }
    else # everythink is okay rename it to lowercase
    {
        mvcmd = "mv " sfile " " dfile
        printf "Renaming %s to %s\n",sfile,dfile
        system(mvcmd)
    }
}

#
# End action, if any, e.g. clean ups
#
END{
}

#
# ./ch.sh: vivek-tech.com to nixcraft.com referance converted using this tool
# See the tool at http://www.nixcraft.com/uniqlinuxfeatures/tools/
#
```

```
#!/bin/bash
#
# up2low : script to convert upercase filename to lowercase in current
# working dir
# Author : Vivek G. Gite <vivek@nixcraft.com>
#
#Copy this file to your bin directory i.e. $HOME/bin as cp rename.awk $HOME/bin
#

AWK_SCRIPT="rename.awk"

#
# change your location here
#
awkspath=$HOME/bin/$AWK_SCRIPT

ls -l > /tmp/file1.$$

tr "[A-Z]" "[a-z]" < /tmp/file1.$$ > /tmp/file2.$$

paste /tmp/file1.$$ /tmp/file2.$$ > /tmp/tmpdb.$$

rm -f /tmp/file1.$$
rm -f /tmp/file2.$$

#
# Make sure awk script exist
#

if [ -f $awkspath ]; then
    awk -f $awkspath /tmp/tmpdb.$$
else
    echo -e "\n$0: Fatal error - $awkspath not found"
    echo -e "\nMake sure \$awkspath is set correctly in $0 script\n"
fi

rm -f /tmp/tmpdb.$$

#
# ./ch.sh: vivek-tech.com to nixcraft.com referance converted using this tool
# See the tool at http://www.nixcraft.com/uniqlinuxfeatures/tools/
#
```

Note: This file is part of Linux Shell Scripting Tutorial, and contains many Linux/Unix definition, miscellaneous concepts and answer to many shell scripts exercise section.

Free

Linux is free.

First ,It's available free of cost (You don't have to pay to use this OS, other Oses like MS-Windows or Commercial version of Unix may cost you money)

Second free means freedom to use Linux, i.e. when you get Linux you will also get source code of Linux, so you can modify OS (Yes OS! Linux OS!!) according to your taste.

It also offers many Free Software applications, programming languages, and development tools etc. Most of the Program/Software/OS are under GNU General Public License (GPL).



Unix Like

Unix is almost 35 year old Os.

In 1964 OS called MULTICS (Multiplexed Information and Computing System) was developed by Bell Labs, MIT & General Electric. But this OS was not the successful one.

Then Ken Thompson (System programmer of Bell Labs) thinks he could do better (In 1991, Linus Torvalds felt he could do better than Minix - History repeats itself.). So Ken Thompson wrote OS on PDP - 7 Computer, assembler and few utilities, this is know as Unix (1969). But this version of Unix is not portable. Then Unix was rewrote in C. Because Unix written in 'C', it is portable. It means Unix can run on verity of Hardware platform (1970-71).

At the same time Unix was started to be distribute to Universities. There students and professor started more experiments on Unix. Because of this Unix gain more popularity, also several new features are added to Unix. Then US govt. & military used Unix for there inter-network (now it is know as INTERNET).

So Unix is Multi-user, Multitasking, Internet-aware Network OS. Linux almost had same Unix Like feature for e.g.

- Like Unix, Linux is also written in C.
- Like Unix, Linux is also the Multi-user/Multitasking/32 or 64 bit Network OS.
- Like Unix, Linux is rich in Development/Programming environment.
- Like Unix, Linux runs on different hardware platform; for e.g.
 - Intel x86 processor (Celeron/PII/PIII/PIV/Old-Pentiums/80386/80486)
 - Macintosh PC's

- Cyrix processor
- AMD processor
- Sun Microsystems Sparc processor
- Alpha Processor (Compaq)



Open Source

Linux is developed under the GNU Public License. This is sometimes referred to as a "copyleft", to distinguish it from a copyright.

Under GPL the source code is available to anyone who wants it, and can be freely modified, developed, and so forth. There are only a few restrictions on the use of the code. If you make changes to the programs, you have to make those changes available to everyone. This basically means you can't take the Linux source code, make a few changes, and then sell your modified version without making the source code available. For more details, please visit [the open-source home page](#).



Common vi editor command list

For this Purpose	Use this vi Command Syntax
To insert new text	esc + i (You have to press 'escape' key then 'i')
To save file	esc + : + w (Press 'escape' key then 'colon' and finally 'w')
To save file with file name (save as)	esc + : + w "filename"
To quit the vi editor	esc + : + q
To quit without saving	esc + : + q!
To save and quit vi editor	esc + : + wq
To search for specified word in forward direction	esc + /word (Press 'escape' key, type /word-to-find, for e.g. to find word ' shri ', type as / shri)
To continue with search	n
To search for specified word in backward direction	esc + ?word (Press 'escape' key, type word-to-find)
To copy the line where cursor is located	esc + yy
To paste the text just deleted or copied at the cursor	esc + p
To delete entire line where cursor is located	esc + dd
To delete word from cursor position	esc + dw
To Find all occurrence of given word and Replace then globally without confirmation	esc + :\$s/word-to-find/word-to-replace/g For. e.g. :\$s/mumbai/pune/g Here word "mumbai" is replace with "pune"

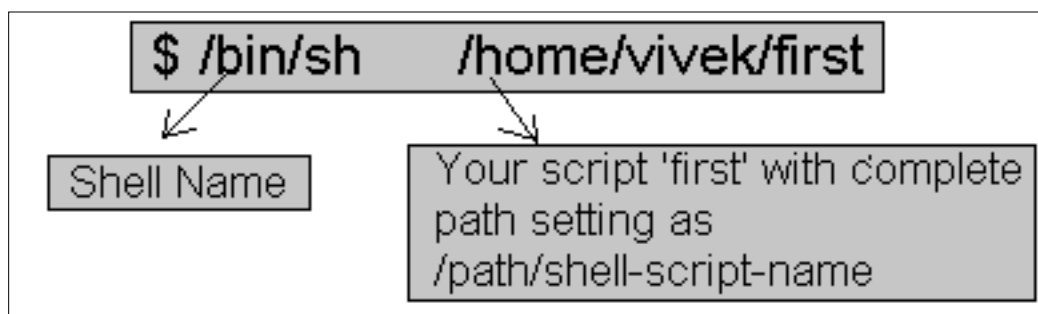
To Find all occurrence of given word and Replace then globally with confirmation	esc + :\$s/word-to-find/word-to-replace/cg
To run shell command like ls, cp or date etc within vi	esc + :!shell-command For e.g. :!pwd



How Shell Locates the file

To run script, you need to have in the same directory where you created your script, if you are in different directory your script will not run (because of path settings), For e.g.. 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 there are two ways first, 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
```



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**) By typing commands like

```
$ bc
```

```
$ cc myprg.c
```

```
$ cal
```

etc, How its possible? All our executables files are installed in directory called /bin and /bin directory is set in your PATH setting, Now when you type name of any command at \$ prompt, what shell do is it first look that command in its internal part (called as internal command, which is part of Shell itself, and always available to execute), if found as internal command shell will execute it, If not found It will look for current directory, if found shell will execute command from current directory, if not found, then Shell will Look PATH setting, and try to find our requested commands executable file in all of the directories mentioned in PATH settings, if found it will execute it, otherwise it will give message "bash: xxxx :command not found", Still there is one question remain can I run my shell script same as these executables?, Yes you can, for this purpose create bin directory in your home directory and then copy

your tested version of shell script to this bin directory. After this you can run you script as executable file without using command like

```
$ /bin/sh /home/vivek/first
```

Command to create you own bin directory.

```
$ cd
$ mkdir bin
$ cp first ~/bin
$ first
```

Each of above commands can be explained as follows:

Each of above command	Explanation
\$ cd	Go to your home directory
\$ mkdir bin	Now created bin directory, to install your own shell script, so that script can be run as independent program or can be accessed from any directory
\$ cp first ~/bin	copy your script 'first' to your bin directory
\$ first	Test whether script is running or not (It will run)



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](#)

ERROR 2 [Read this](#)

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"
```




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/lst"` 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/lst
```

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/lst`. 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
```



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.



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.



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
```



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

<code>\033</code>	Escape character
[Start of CSI
34	34 is <u>parameter</u>
m	<u>m</u> 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	<code>echo -e "\033[h"</code>
l	Clears the ANSI mode	<code>echo -e "\033[l"</code>
m	Useful to show characters in different colors or effects such as BOLD and Blink, see below for parameter taken by m.	<code>echo -e "\033[35m Hello World"</code>
q	Turns keyboard num lock, caps lock, scroll lock LED on or off, see below.	<code>echo -e "\033[2q"</code>
s	Stores the current cursor x,y position (col , row position) and attributes	<code>echo -e "\033[7s"</code>
u	Restores cursor position and attributes	<code>echo -e "\033[8u"</code>

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	<code>\$ echo -e "I am \033[1m BOLD \033[0m Person"</code> I am BOLD Person Prints BOLD word in bold intensity and next ANSI Sequence remove bold effect (<code>\033[0m</code>)

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.](#)

[Click here to see example of m command.](#)

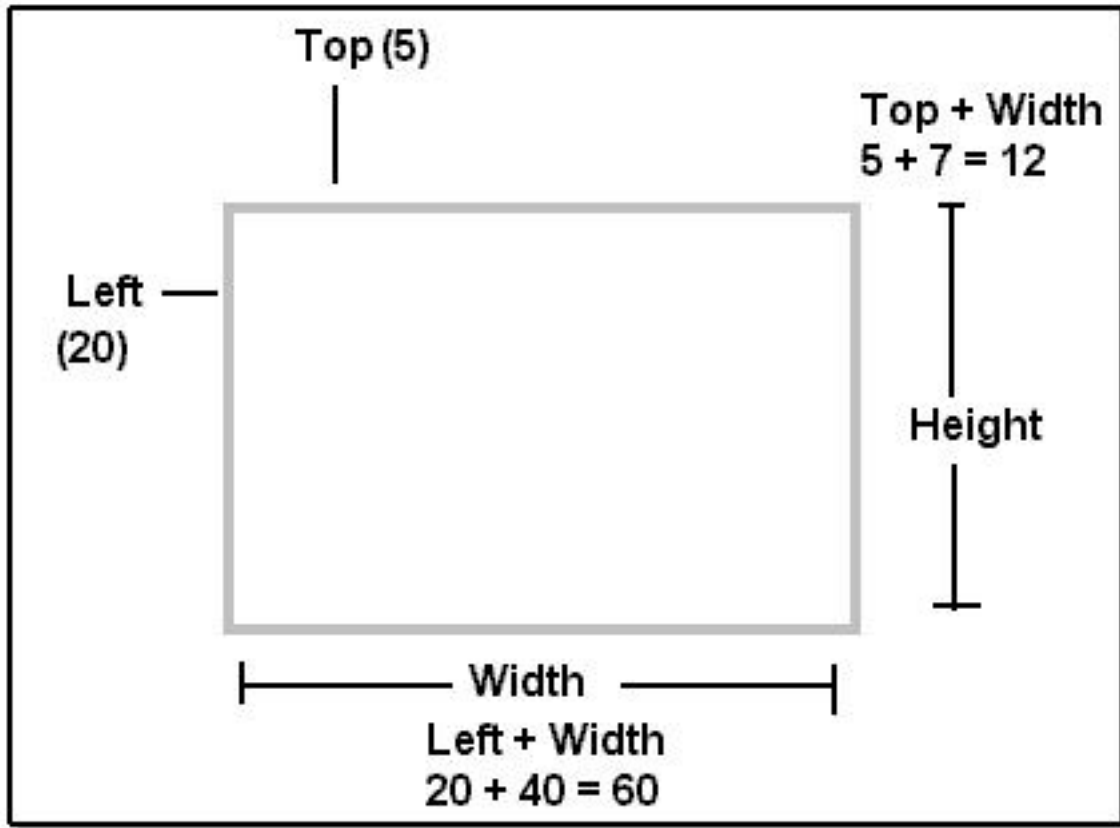
[Click here to see example of s and u command.](#)

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.



Shell Built in Variables

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

[See example of `\$@` and `\$*` variable.](#)



```
dialog --title "Linux Dialog Utility Infobox" --backtitle "Linux Shell Script\  
Tutorial" --infobox "This is dialog box called infobox, which is used \  
to show some information on screen, Thanks to Savio Lam and\  
Stuart Herbert to give us this utility. Press any key. . . " 7 50 ; read
```

```
#  
# ./ch.sh: vivek-tech.com to nixcraft.com referance converted using this tool  
# See the tool at http://www.nixcraft.com/uniqlinuxfeatures/tools/  
#
```



```
dialog --title "Linux Dialog Utility Msgbox" --backtitle "Linux Shell Script\  
Tutorial" --msgbox "This is dialog box called msgbox, which is used \  
to show some information on screen which has also Ok button, Thanks to Savio Lam\  
and Stuart Herbert to give us this utility. Press any key. . . " 9 50
```

```
#  
# ./ch.sh: vivek-tech.com to nixcraft.com referance converted using this tool  
# See the tool at http://www.nixcraft.com/uniqlinuxfeatures/tools/  
#
```

```
dialog --title "Alert : Delete File" --backtitle "Linux Shell Script\
Tutorial" --yesno "\nDo you want to delete '/usr/letters/jobapplication'\
file" 7 60
sel=$?
case $sel in
    0) echo "User select to delete file";;
    1) echo "User select not to delete file";;
    255) echo "Canceled by user by pressing [ESC] key";;
esac

#
# ./ch.sh: vivek-tech.com to nixcraft.com referance converted using this tool
# See the tool at http://www.nixcraft.com/uniqlinuxfeatures/tools/
#
```

```
dialog --title "Inputbox - To take input from you" --backtitle "Linux Shell\  
Script Tutorial" --inputbox "Enter your name please" 8 60 2>/tmp/input.$$
```

```
sel=$?
```

```
na=`cat /tmp/input.$$`
```

```
case $sel in
```

```
0) echo "Hello $na" ;;
```

```
1) echo "Cancel is Press" ;;
```

```
255) echo "[ESCAPE] key pressed" ;;
```

```
esac
```

```
rm -f /tmp/input.$$
```

```
#
```

```
# ./ch.sh: vivek-tech.com to nixcraft.com referance converted using this tool
```

```
# See the tool at http://www.nixcraft.com/uniqlinuxfeatures/tools/
```

```
#
```

```
#
#How to create small menu using dialog
#
dialog --backtitle "Linux Shell Script Tutorial " --title "Main \
Menu" --menu "Move using [UP] [DOWN],[Enter] to \
Select " 15 50 3 \
Date/time      "Shows Date and Time" \
Calendar       "To see calendar " \
Editor         "To start vi editor " 2>/tmp/menuitem.$$

menuitem=`cat /tmp/menuitem.$$`

opt=$?

case $menuitem in
    Date/time) date;;
    Calendar) cal;;
    Editor) vi;;
esac

#
# ./ch.sh: vivek-tech.com to nixcraft.com referance converted using this tool
# See the tool at http://www.nixcraft.com/uniqlinuxfeatures/tools/
#
```

11	Vivek
12	Renuka
13	Prakash
14	Ashish
15	Rani

11	67
12	55
13	96
14	36
15	67

egg	order	4
cacke	good	10
cheese	okay	4
pen	good	12
Floppy	good	5

India's milk is good.
tea Red-Lable is good.
tea is better than the coffee.

Hello I am vivek

12333

12333

welcome

to

sai computer academy, a'bad.

what still I remeber that name.

oaky! how are u luser?

what still I remeber that name.

hello world!
cartoons are good
especially toon like tom (cat)
what
the number one song
12221
they love us
I too

Hello World.

This is vivek from Poona.

I love linux.

It is different from all other Os

My brother Vikrant also loves linux who also loves unix.

He currently learn linux.

Linux is coool.

Linux is now 10 years old.

Next year linux will be 11 year old.

Rani my sister never uses Linux

She only loves to play games and nothing else.

Do you know?

. (DOT) is special command of linux.

Okay! I will stop.

1. Pen	5	20.00
2. Pencil	10	2.00
3. Rubber	3	3.50
4. Cock	2	45.50

```
{  
  print $1 " + " $2 " = " $1 + $2  
  print $1 " - " $2 " = " $1 - $2  
  print $1 " / " $2 " = " $1 / $2  
  print $1 " x " $2 " = " $1 * $2  
  print $1 " mod " $2 " = " $1 % $2  
}
```

```
{  
  no1 = $1  
  no2 = $2  
  ans = $1 + $2  
  print no1 " + " no2 " = " ans  
}
```

```
{  
  total = $3 * $4  
  recno = $1  
  item = $2  
  print recno item " Rs." total  
}
```

```
BEGIN {
    print "-----"
    print "Bill for the 4-March-2001. "
    print "By Vivek G Gite.          "
    print "-----"
}
{
    total = $3 * $4
    recno = $1
    item = $2
    gtotal += total
    print recno item " Rs." total
}
END {
    print "-----"
    print "Total Rs." gtotal
    print "====="
```



```
BEGIN {
    printf "Bill for the 4-March-2001.\n"
    printf "By Vivek G Gite.\n"
    printf "-----\n"
}
{
    total = $3 * $4
    recno = $1
    item = $2
    gtotal += total
    printf "%d %s Rs.%f\n", recno, item, total
    #printf "%2d %-10s Rs.%7.2f\n", recno, item, total
}
END {
    printf "-----\n"
    printf "Total Rs. %f\n" ,gtotal
    #printf "\tTotal Rs. %7.2f\n" ,gtotal
    printf "=====\n"
}
```

```
BEGIN {
    printf "Bill for the 4-March-2001.\n"
    printf "By Vivek G Gite.\n"
    printf "-----\n"
}
{
    total = $3 * $4
    recno = $1
    item = $2
    gtotal += total
    printf "%2d %-10s Rs.%.7.2f\n", recno, item, total
}
END {
    printf "-----\n"
    printf "\tTotal Rs. %6.2f\n" ,gtotal
    printf "=====\n"
}
```

```
BEGIN {
  myprompt = "(To Stop press CTRL+D) > "
  printf "Welcome to MyAddition calculation awk program v0.1\n"
  printf "%s" ,myprompt
}

{
  no1 = $1
  op = $2
  no2 = $3
  ans = 0

  if ( op == "+" )
  {
    ans = $1 + $3
    printf "%d %c %d = %d\n" ,no1,op,no2,ans
    printf "%s" ,myprompt
  }
  else
  {
    printf "Opps!Error I only know how to add.\nSyntax: number1 + number2\n"
    printf "%s" ,myprompt
  }
}

END {
  printf "\nGoodbuy %s\n" , ENVIRON["USER"]
}
```

```
BEGIN{
    printf "Press ENTER to continue with for loop example from LSST v1.05r3\n"
}
{
    sum = 0
    i = 1
    for (i=1; i<=10; i++)
    {
        sum += i; # sum = sum + i
    }
    printf "Sum for 1 to 10 numbers = %d \nGoodbuy!\n\n", sum
    exit 1
}
```

/home/vivek/awks/temp/file1	/home/vivek/final
/home/vivek/awks/temp/file2	/home/vivek/final
/home/vivek/awks/temp/file3	/home/vivek/final
/home/vivek/awks/temp/file4	/home/vivek/final

```
{
dcmd = "rm " $1
if ( system(dcmd) != 0 )
    printf "rm command not successful\n"
else
    printf "rm command is successful and %s file is removed \n", $1
}
```

```
BEGIN {  
    printf "Your name please:"  
    getline na < "-"  
    printf "%s your age please:",na  
    getline age < "-"  
    print "Hello " na, ", next year you will be " age + 1  
}
```

```
BEGIN{  
  "date" | getline  
  print $0  
}
```



```
BEGIN{  
  "date" | getline today  
  print today  
}
```

```
#
#temp2final1.awk: Version 2
#Linux Shell Scripting Tutorial v1.05, March 2001
#
#Author: Vivek G Gite
#
#
#This version checks for source and destination file first
#then copy the file. If file already exist it will ask confirmation.
#
#
#
BEGIN{
}

#
# main logic is here
#
{
    sfile = $1
    dfile = $2
    issexist = "[ -e " $1 " ]"
    isdexist = "[ -e " $2 " ]"
    cpcmd = "cp " $1 " " $2
    printf "Coping %s to %s\n",sfile,dfile
    if( ( system(issexist) ) != 0 )
    {
        printf "Skipking \"%s\", does not exist\n",sfile
        next # read next record
    }

    if ( ( system(isdexist) ) == 0)
    {
        printf "\"%s\", exist overwrite(y/N)?", sfile
        getline ans < "-"
        if( ans == "y" || ans == "Y")
            system(cpcmd)
    }
    else
        system(cpcmd)
}

#
# End action, if any, e.g. clean ups
#
END{
}

#
# ./ch.sh: vivek-tech.com to nixcraft.com referance converted using this tool
# See the tool at http://www.nixcraft.com/uniqlinuxfeatures/tools/
#
```

Hello World.
This is vivek from Poona.

I love linux.
It is different from all other Os

My brother Vikrant also loves linux who also loves unix.
He currently learn linux.
Linux is coool.

Linux is now 10 years old.
Next year linux will be 11 year old.

Rani my sister never uses Linux
She only loves to play games and nothing else.

Do you know?
. (DOT) is special command of linux.

Okay! I will stop.

I care for you and
Vivek care for.
1224
welcome
6888
linux liux
linux is linux

1i\
Price of all items changes from 1st-April-2001
/Pen/s/20.00/19.5/
/Pencil/s/2.00/2.60/
/Rubber/s/3.50/4.25/
/Cock/s/45.50/51.00/

Today's date is 5-12-01 i.e. 5-Dec-2001

My brother Vivkran was born on 5-Dec-70

My birthdate : April 5, 00

Renu my sister was born on 6-1-74

Numbers fun Binary numbers

1001

100001

10001

1000000001

10101010

10101010

Okay Linux is just like a star.

Star brings good things to life

When I was little kid

I love to see star, my mother says star are Gods Gift to Us!

Is there any relation between star and Linux

```
/^\*\{2,3\}$/,/^\*\{2,3\}$/{  
  /^\$/d  
  s/Linux/Linux-Unix/  
}
```

Welcome to world of sed what sed is?

I don't know what sed is but I think

Rani knows what sed is

Name of Friend	DOB	Hobby	Phone #
V.K. Rajopadhey 5/22,Stree 4, A'bad,MH, INDIA.	5/12/73	Food, Music	98220-5678
A.G. Gite 22, MIDC, Mumbai,MH, INDIA.	15/6/72	Computers, Book Reading	98220-3333
M.M. Kale 6/21,Silver Estate, A'bad,MH, INDIA.	2/1/71	Food, Drinks, Lifestyle	98220-6823
R.K. Joshi Flat No.9, Pushpa Towers, Pune,MH, INDIA.	9/10/70	Colletion of Old coins	98220-6877
N.K. Kulkarni Sector 20, Padmavti, Pune,MH, INDIA.	1/2/74	Computer Games	98220-9888
