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

nixCraft Technologies
*nix Support | Implementation | Education | Development

(Formally know as 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
Coping 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. suppiled as command line args
Calculating average of given numbers on command line args
Finding out biggest number from given three nos suppiled as command line args
Shell script to implement getopts 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 Entery 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

About this Document

The latest copy of this document can always be obtained from: http://www.nixcraft.com/docs/

Last updated Linux Shell Scripting Tutorial v1.05 (LSST) - on Wednesday, April 24, 2002.

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 enplaning 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 scripting.

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

Also not that currently this tutorial is also translated into some other foreign language(s); if you are interested to read it in other language the the English then please visit http://www.nixcraft.com/uniqlinuxfeatures/lsst/.

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 and off course bugs (errors) of this tutorial.

What Linux is?

- Free
- Unix Like
- Open Source
- Network operating system

Who developed the Linux?

In 1991, Linus Torvalds studding 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/	redhat®
SuSE Linux: http://www.suse.com/	SuSE
Mandrake Linux: http://www.mandrakesoft.com/	MandrakeSoft™
Caldera Linux: http://www.calderasystems.com/	CALDERA systems
Debian GNU/Linux: http://www.debian.org/	debian
Slackware Linux: http://www.slackware.com/	slackware linux

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 flavor) or if you are in Pune, India please visit the our sponsor 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 | Home | Next
Who developed the Linux? | Up | How to Install Linux

How to Install Linux ?

Please visit the http://www.nixcraft.com/lessbs/ for Quick Visual Installation Guide for Red Hat Linux version 6.2 and 7.2.

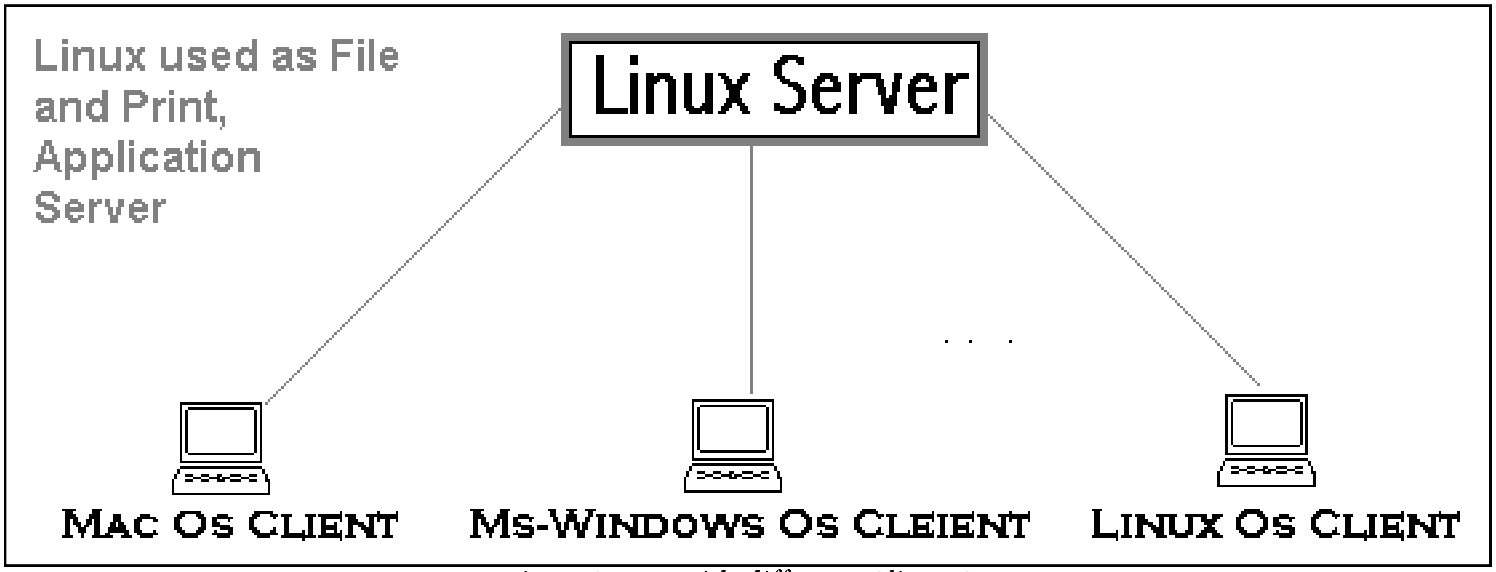
Where I can use Linux?

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

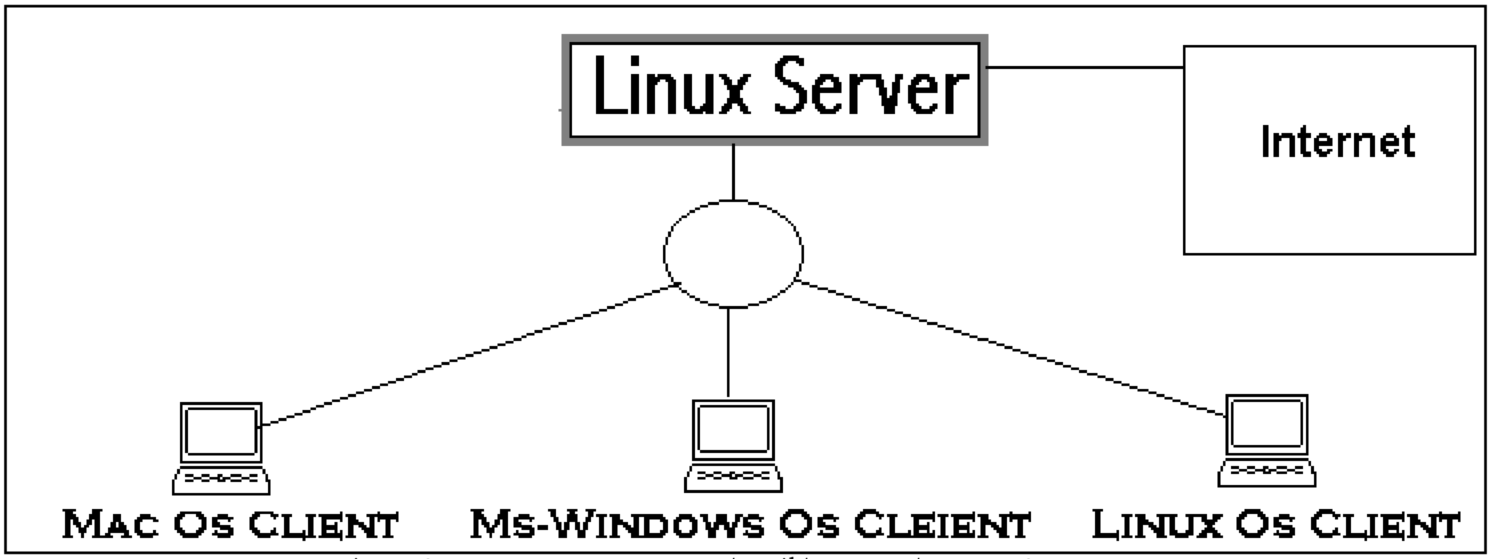
Linux offers all of the above characteristics plus its Open Source and Free OS. So Linux can be used as:

(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.



Linux Server with different Client Os

(B) Linux sever cab 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 Home Next
How to Install Linux Up What Kernel Is?

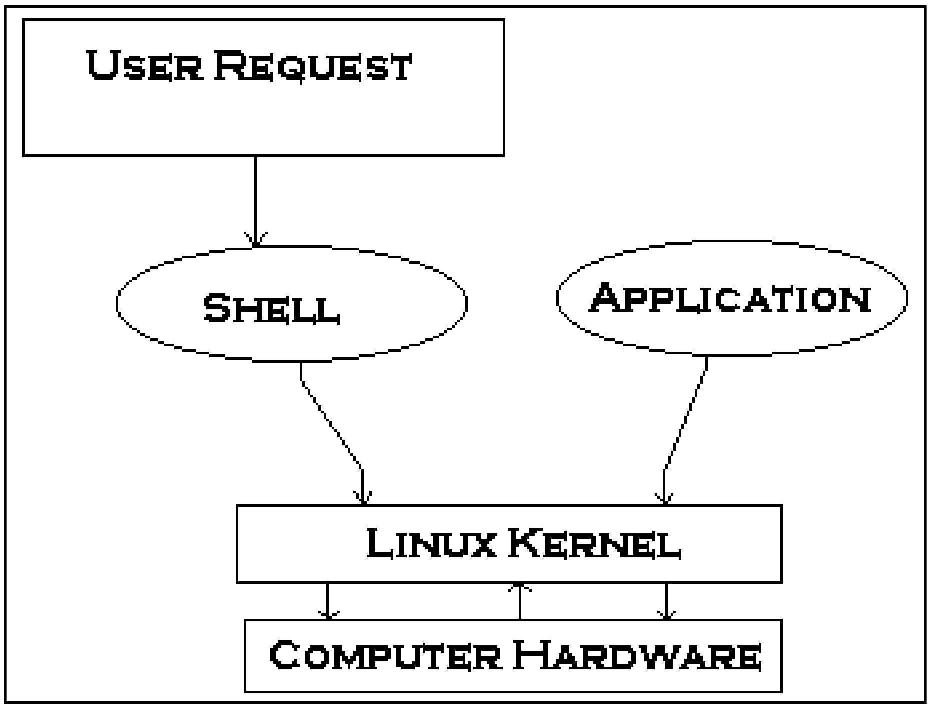
What Kernel Is?

Kernel is hart 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

Prev	Home	Next
Where I can use Linux?	Up	What is Linux Shell?

What is Linux Shell ?

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

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

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

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

Several shell available with Linux including:

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

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

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

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

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

Tip: To find your current shell type following command

```
$ echo $SHELL
```

Prev Home Next
What Kernel Is? Up 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 introduce to shell programming, how to write script, execute them etc. We will getting started with writing small shell script, that will print "Knowledge is Power" on screen. Before starting with this you should know

- How to use text editor such as vi, see the common vi command for more information.
- Basic command navigation

How to write shell script

Following steps are required to write shell script:

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

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

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

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

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

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

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

Example:
$. foo

Now you are ready to write first shell script that will print "Knowledge is Power" on screen. See the common vi command list , if you are new to vi.

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

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

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

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

Script Command(s)	Meaning
$ vi first	Start vi editor
# # My first shell script #	# followed by any text is considered as comment. Comment gives more information about script, logical explanation about shell script. *Syntax:* # comment-text
clear	clear the screen
echo "Knowledge is Power"	To print message or value of variables on screen, we use echo command, general form of echo command is as follows *syntax:* echo "Message"

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 the it's output.

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

Future Point: At the end why statement exit 0 is used? See exit status for more information.

Variables in Shell

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

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

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

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

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

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

OR

(b)$ echo HOME

Which of the above command is correct & why? Click here for answer.

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

Prev	Home	Next
How to write shell script	Up	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.

Prev | Home | Next
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

Prev	Home	Next
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 Home Next
echo Command Up More about Quotes

More about Quotes

There are three types of quotes

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

Example:
$ echo "Today is date"
Can't print message with today's date.
$ echo "Today is `date`".
It will print today's date as, Today is Tue Jan ....,Can you see that the `date` statement uses back quote?

Exit Status

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

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

This value is know as *Exit Status*.

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

For e.g. (This example assumes that unknow1file doest not exist on your hard drive)
$ rm unknow1file
It will show error as follows
rm: cannot remove `unkowm1file': No such file or directory
and after that if you give command
$ echo $?
it will print nonzero value to indicate error. Now give command
$ ls
$ echo $?
It will print 0 to indicate command is successful.

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

$ echo Welcome
$ echo $?

$ wildwest canwork?
$ echo $?

$ date

```
$ echo $?

$ echon $?
$ echo $?
```

$? useful variable, want to know more such Linux variables click here to explore them!

The read Statement

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

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

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

Run it as follows:

```
$ chmod 755 sayH
$ ./sayH
Your first name please: vivek
Hello vivek, Lets be friend!
```

Wild cards (Filename Shorthand or meta Characters)

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

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

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

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

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

But

```
$ ls /bin/[!a-o]
$ ls /bin/[^a-o]
```

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

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

Prev Home Next
The read Statement Up 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 Home Next
More commands on one command line Up Why Command Line arguments required

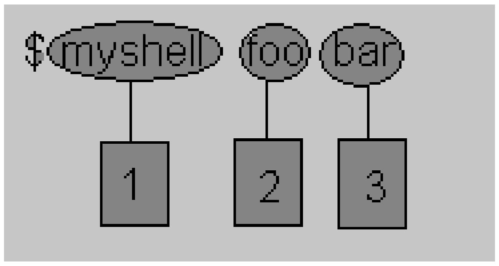
Prev Next

Why Command Line arguments required

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

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

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



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 $#"
echo "$0 is script name"
echo "$1 is first argument"
echo "$2 is second argument"
echo "All of them are :- $* or $@"
```

Run it as follows

Set execute permission as follows:
$ chmod 755 demo

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

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

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

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

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

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 command gives output on screen or take input from keyboard, but in Linux (and in other OSs also) it's possible to send output to file or to read input from file.

For e.g.
$ ls command gives output to screen; to send output to file of ls command give command

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

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

(1) > Redirector Symbol
Syntax:
Linux-command > filename
To output Linux-commands result (output of command or shell script) to file. Note that if file already exist, it will be overwritten else new file is created. For e.g. To send output of ls command give
$ ls > myfiles
Now if 'myfiles' file exist in your current directory it will be overwritten without any type of warning.

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

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

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
```

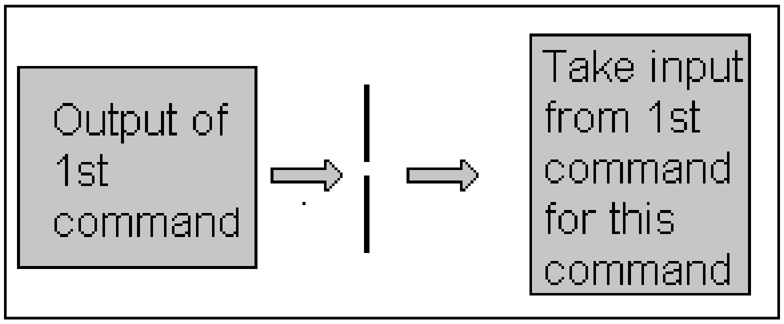
tr command is used to translate all lower case characters to upper-case letters. It take input from sname file, and tr's output is redirected to cap_names file.

Future Point : Try following command and find out most important point:
```
$ sort > new_sorted_names < sname
$ cat new_sorted_names
```

Prev Home Next
Why Command Line arguments required Up 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

Examles:

Command using Pipes	Meaning or Use of Pipes
$ ls | more	Output of ls command is given as input to more command So that output is printed one screen full page at a time.
$ who | sort	Output of who command is given as input to sort command So that it will print sorted list of users
$ who | sort > user_list	Same as above except output of sort is send to (redirected) user_list file

$ who | wc -l	Output of who command is given as input to wc command So that it will number of user who logon to system
$ ls -l | wc -l	Output of ls command is given as input to wc command So that it will print number of files in current directory.
$ who | grep raju	Output of who command is given as input to grep command So that it will print if particular user name if he is logon or nothing is printed (To see particular user is logon or not)

Prev Home Next
Redirection of Standard output/input i.e. Input - Output redirection Up 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 uniqs 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 equivalnt 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 bashs structured language constricts such as:

- Decision making
- Loops

Is there any difference making decision in Real life and with Computers? Well real life decision are quit 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 | Home | Next
Linux Command(s) Related with Process | Up | 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.

Expreession 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 referance 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.

---

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 > ispostive
#!/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 ispostive

$ ispostive 5
5 number is positive

$ispostive -45
Nothing is printed

$ispostive
./ispostive: 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 ./ispostive: 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 Home Next
Decision making in shell script (i.e. if command) Up 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 of 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* constuct 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 *if*s as follows also:
Syntax:

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

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 | Home | Next
if...else...fi | Up | 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 equel 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
$ ./nestefor.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
```

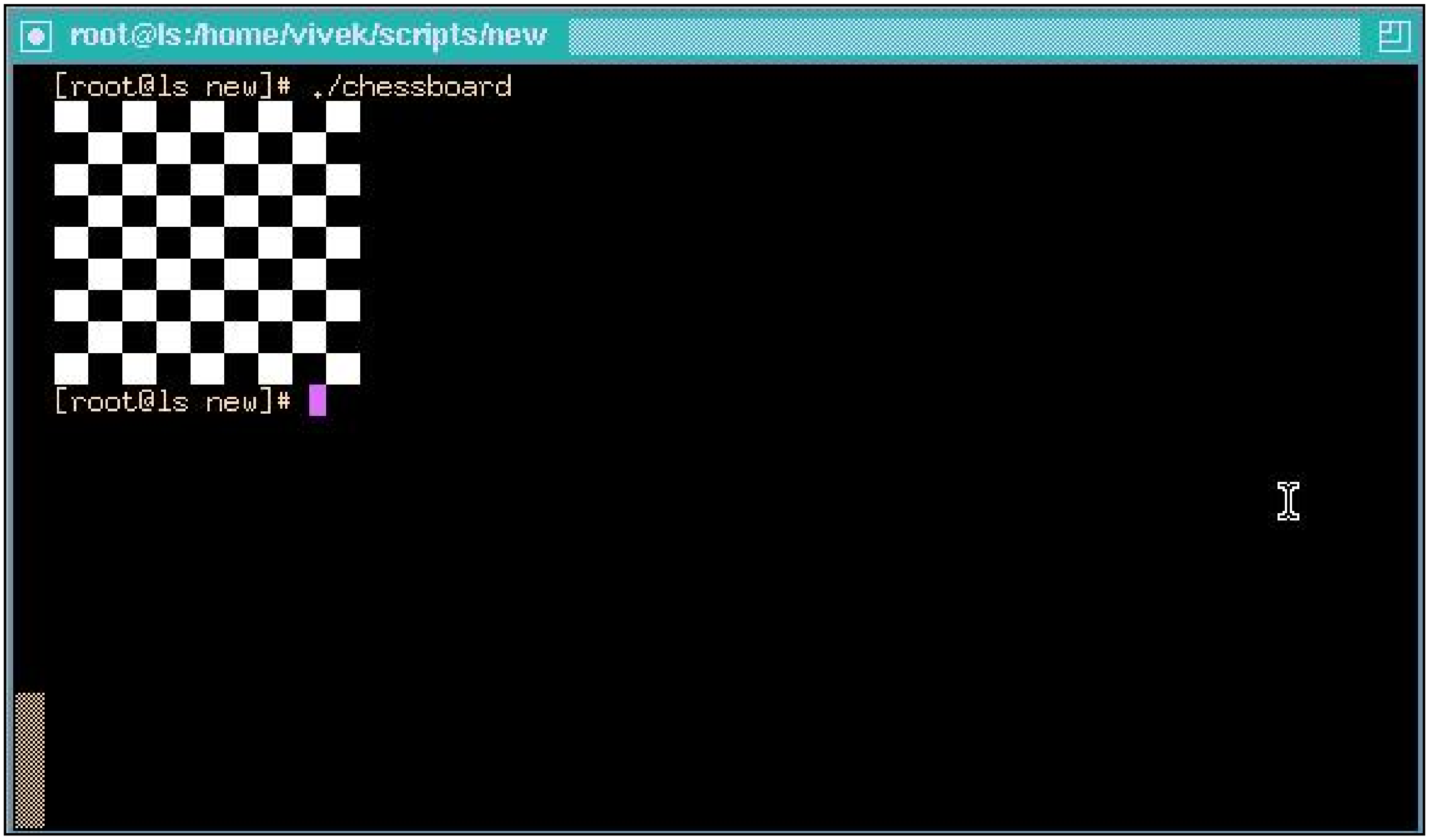
Here, for each value of i the inner loop is cycled through 5 times, with the varible j taking values from 1 to 5. The inner for loop terminates when the value of j exceeds 5, and the outer loop terminets when the value of i exceeds 5.

Following script is quite intresting, 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 produec the output as follows:



```
[root@ls new]# ./chessboard
[root@ls new]#
```

Above shell script cab 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 terminets 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 varible 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 posiotion print the white colour block (using echo -e -n "\033[47m " statement); otherwise for odd postion print the black colour box (using echo -e -n "\033[40m " statement). This statements are responsible to print entier chess board on screen with alternet 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 terminted by printing the chess board.

Exercise
Try to understand the shell scripts (for loops) shown in exercise chapter.

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.

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 Home Next
while loop Up 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 | Home | Next
The case Statement | Up | 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

/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?

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 varible 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 varible
$ /bin/bash	Now load second shell in memory (Old shell's variable is accessed from second shell, *if they are exported*)
$ 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 Home Next
/dev/null - to send unwanted output of program Up 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)

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: (Assemums 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 onsider another example by writing shell script as follows:

```
$ cat > demoscr
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 demoscr
$ ./demoscr
Error : Number are not supplied
Usage : ./demoscr 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 redirect 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	Home	Next
Conditional execution i.e. && and ||	Up	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 is know as *function*.

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 automat 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 Home Next
I/O Redirection and file descriptors Up 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 "Opps!!! 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

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 "======================="	Show menu on screen with menu items
echo -n "Enter your menu choice [1-5]: "	Ask user to enter menu item number
read yourch	Read menu item number from user

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
	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

done	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 Home Next
Functions Up 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 in 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
```

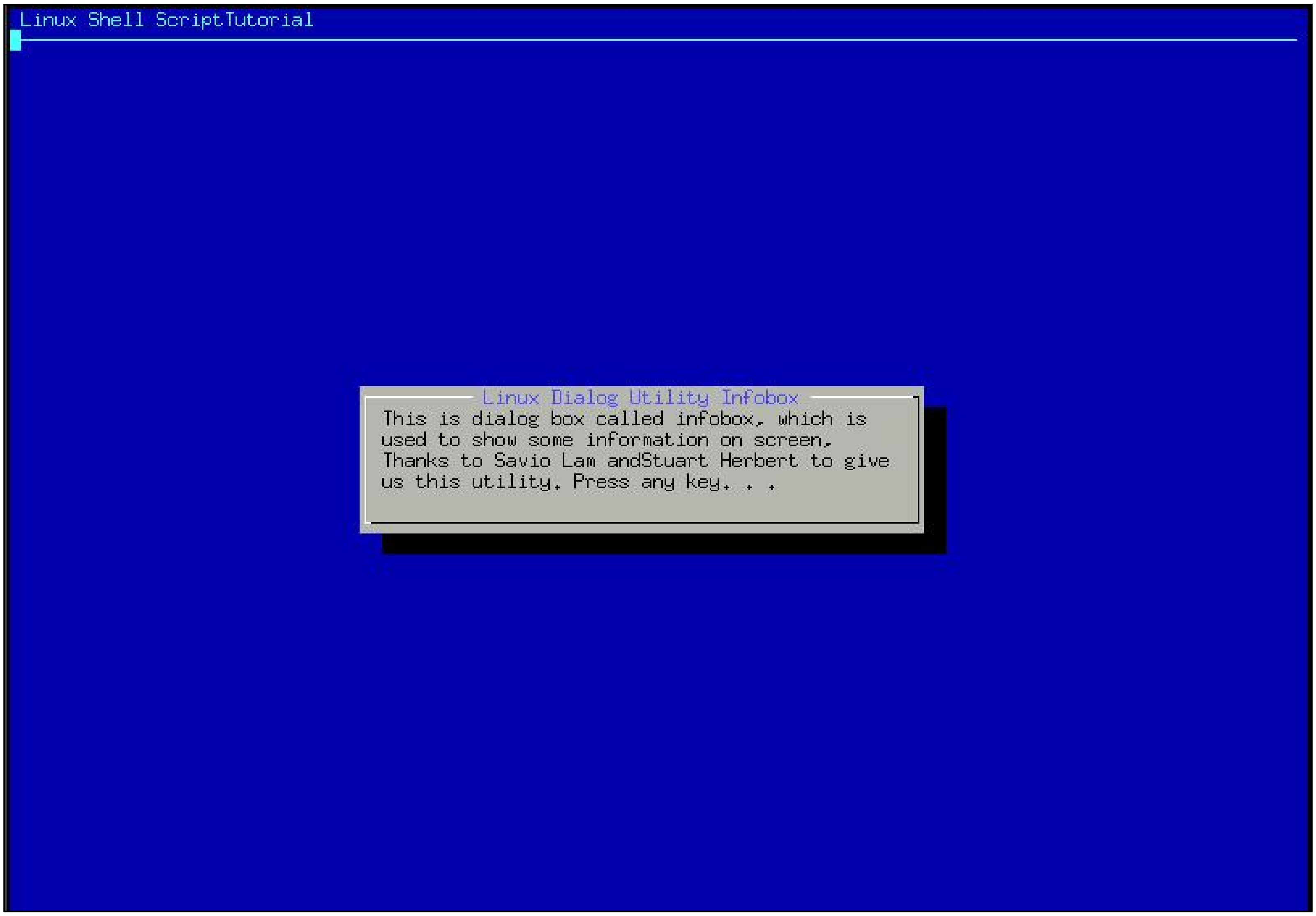
For Red Hat Linux 7.2 user install the dialog utility as follows (First insert Red Hat Linux 7.2 # 1 CD into CDROM drive)

```
# 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 > dia1
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 dia1
$ ./dia1



```
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 andStuart 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:

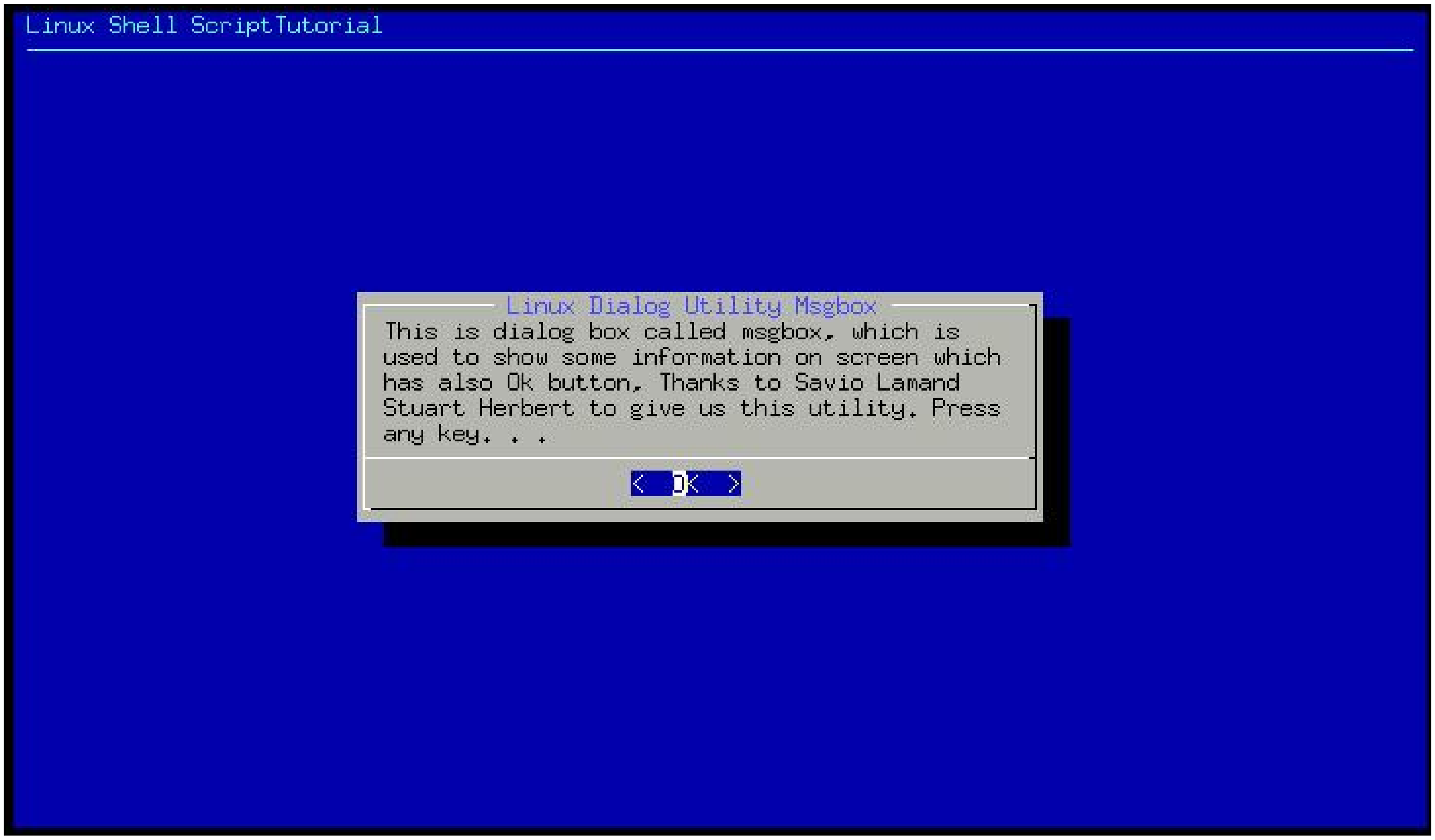
dialog --title {title} --backtitle {backtitle} {Box options}
where Box options can be any one of following
--yesno {text} {height} {width}
--msgbox {text} {height} {width}
--infobox {text} {height} {width}
--inputbox {text} {height} {width} [{init}]
--textbox {file} {height} {width}
--menu {text} {height}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

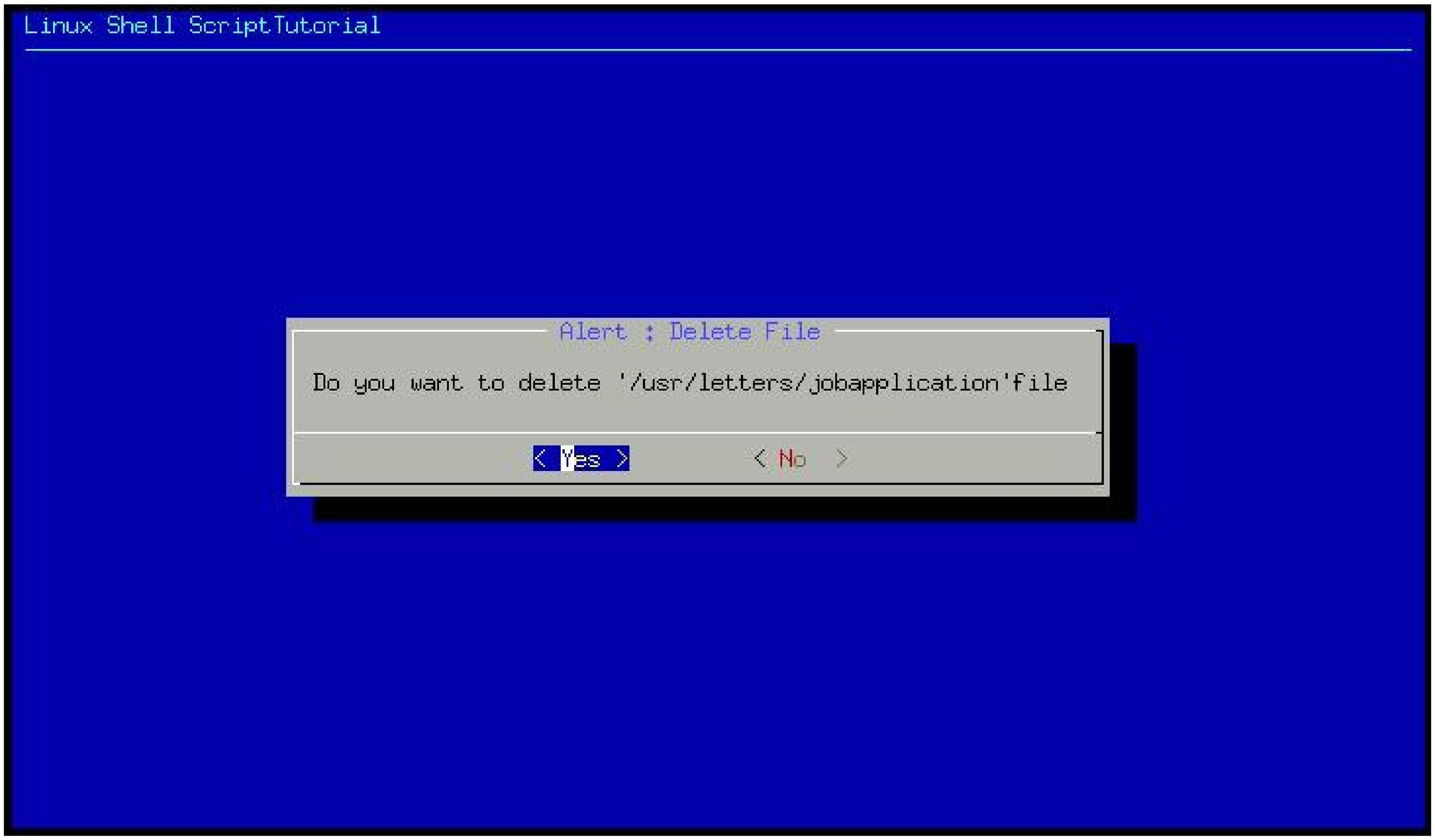


```
Linux Shell ScriptTutorial

              Linux Dialog Utility Msgbox
 This is dialog box called msgbox, which is
 used to show some information on screen which
 has also Ok button, Thanks to Savio Lamand
 Stuart Herbert to give us this utility. Press
 any key. . .

                     < OK >
```

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



```
Linux Shell ScriptTutorial

Alert : Delete File
Do you want to delete '/usr/letters/jobapplication'file

< Yes >          < No >
```

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

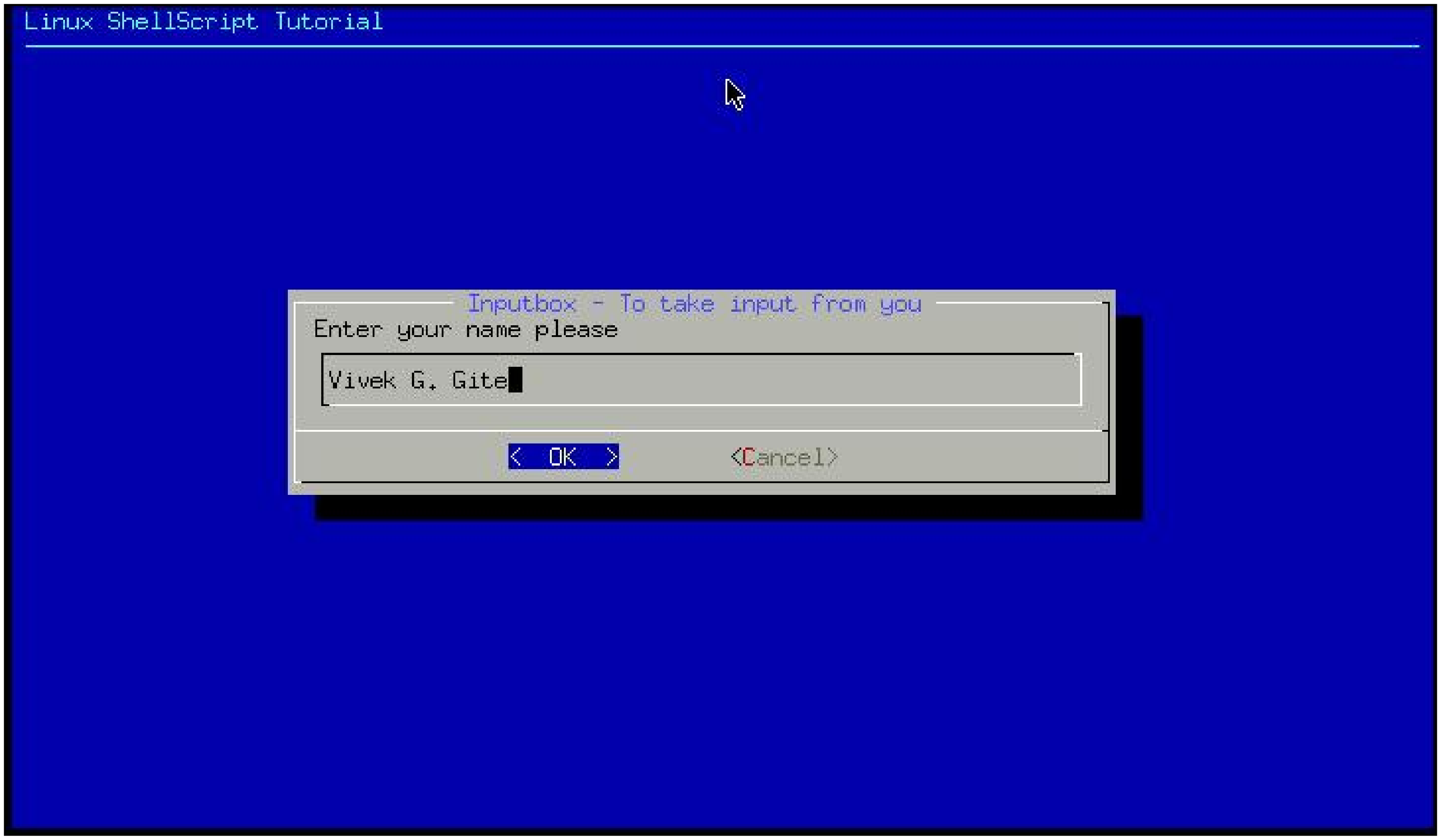
Statement	Meaning
sel=$?	Get exit status of dialog utility
case $sel in 0) echo "You select to delete file";; 1) echo "You select not to delete file";; 255) echo "Canceled by you by pressing [Escape] key";; esac	Now take action according to exit status of dialog utility, if exit status is 0 , delete file, if exit status is 1 do not delete file and if exit status is 255, means Escape key is pressed.

Prev Home Next
Message Box (msgbox) using dialog utility Up 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


```
Linux ShellScript Tutorial

Inputbox - To take input from you
Enter your name please
Vivek G. Gite
```

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.$$* at the end of dialog command, which means send screen output to file called */tmp/input.$$*, letter 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 — Home — Next
Confirmation Box (yesno box) using dialog utility — Up — User Interface using dialog Utility - Putting it all together

User Interface using dialog Utility - Putting it all together

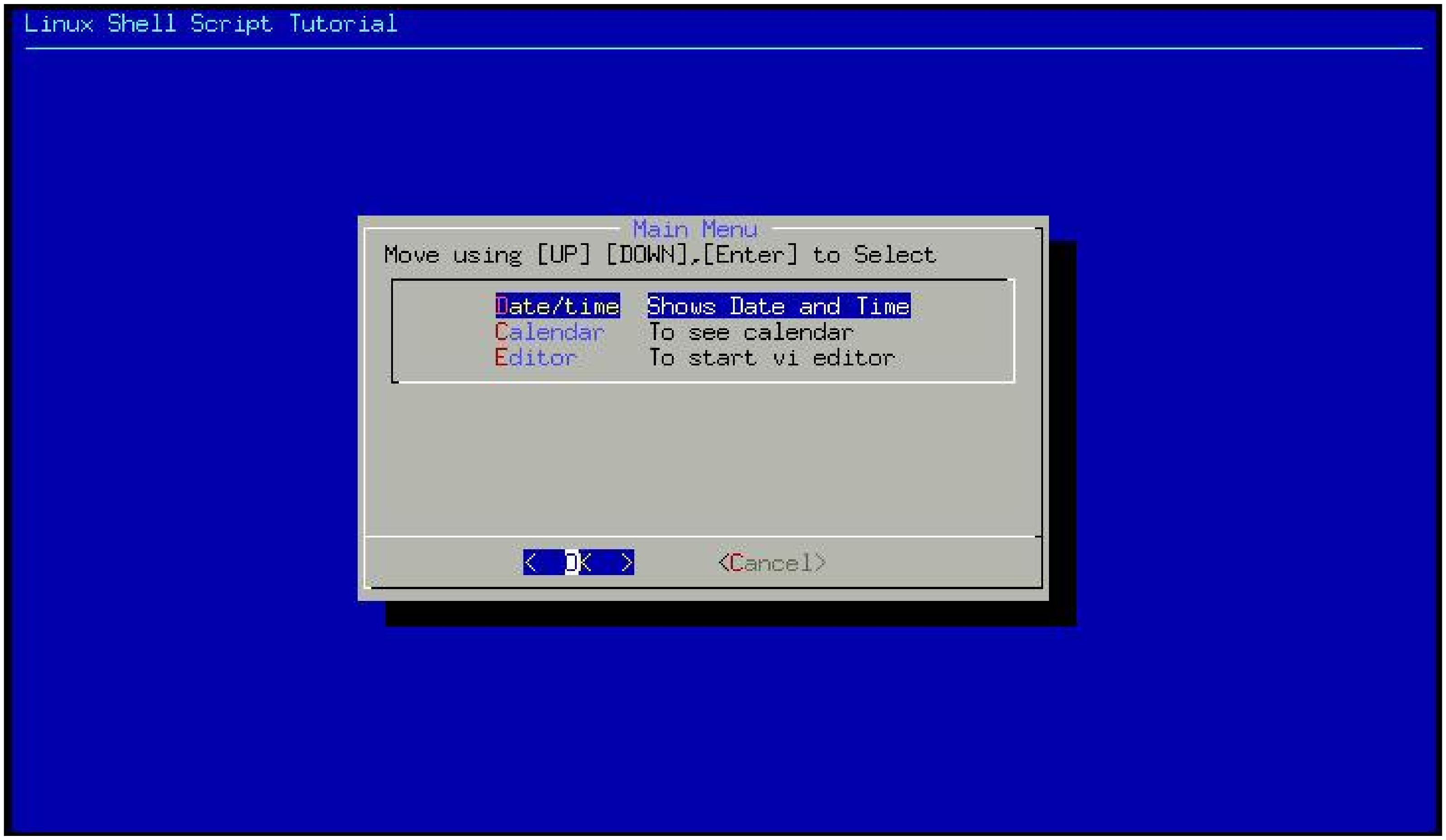
Its time to write script to create menus using dialog utility, following are menu items
Date/time
Calendar
Editor
and action for each menu-item is follows :

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
```



```
Move using [UP] [DOWN],[Enter] to Select

      Date/time    Shows Date and Time
      Calendar     To see calendar
      Editor       To start vi editor
```

--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 show 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 *tag1* (i.e. Date/time) and description for menu item called as *item1* (i.e. "Shows Date and Time")
Calendar "To see calendar "	First menu item called as *tag2* (i.e. Calendar) and description for menu item called as *item2* (i.e. "To see calendar")
Editor "To start vi editor "	First menu item called as *tag3* (i.e. Editor) and description for menu item called as *item3* (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.

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 exit, 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 exit, 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 # end_while
}
#
# Set trap to for CTRL+C interrupt i.e. Install our error handler
# When occurs it first it 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 Home Next
User Interface using dialog Utility - Putting it all together Up 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

while ["$1"]
do
	Begins the while loop; continue the while loop as long as script reads the all command line option

if ["$1" = "-b"]; then
ob="$2"
	Now start to parse the command line (args) option using if command our script understands the -b and -n options only all other option are invalid. If option is -b then stores the value of second command line arg to variable ob (i.e. if arg is -b 16 then store the 16 to ob)

case $ob in 16) basesystem="Hex";; 8) basesystem="Oct";; 2) basesystem="bin";; *) basesystem="Unknown";; esac	For easy understanding of conversion we store the respective number base systems corresponding string to basesystem variable. If base system is 16 then store the Hex to basesystem and so on. This is done using case statement.

shift 2	Once first two command line options (args) are read, we need next two command line option (args). shift 2 will moves the current values stored in the positional parameters (command line args) to the left two position.

elif ["$1" = "-n"]
then
num="$2"
shift 2
	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.

else
echo "Program $0 does not recognize option $1"
exit 1
fi
	If command line option is not -n or -b then print the error "*Program ./convert does not recognize option xx*" on screen and terminates the shell script using *exit 1* statement.

done	End of loop as we read all the valid command line option/args.
output=\`echo "obase=$ob;ibase=10; $num;" \| BC\` echo "$num Decimal number = $output in $basesystem number system (base=$ob)"	Now convert the given number to given number system using BC Show the converted number on screen.

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 - getopts. Next section shows the use of getopts command. You still need the shift command in conjunction with getopts and for other shell scripting work.

Prev Home Next
trap command Up 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 {optsring} {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."

Examlpe:
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 [ $# -lt 1 ]; then
  help_ani
fi
while getopts n:a:s:w:d opt
do
  case "$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 | Home | Next
The shift command | Up | 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.

Prev Home Next
getopts command Up Preparing for Quick Tour of essential utilities

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

Sr.No	Name
11	Vivek
12	Renuka
13	Prakash
14	Ashish
15	Rani

smark

Sr.No	Mark
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](#) [Home](#) [Next](#)
Preparing for Quick Tour of essential utilities [Up](#) 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 Home Next
The join utility Up 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,

/good/	Is the pattern used for selecting lines from file.
{print $3}	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?)
inventory	File which is used by awk utility which is use as input for awk utility.

Use of awk utility:
To manipulate data.

Prev Home Next
Translating range of characters using tr utility Up sed utility - Editing file without using editor

sed utility - Editing file without using editor

For this part of tutorial create data file as follows

teaormilk

```
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' teaormilk  > /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 Home Next
sed utility - Editing file without using editor Up 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 chpater 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 know as metacharacters*".

expressions defined as:
"*Combination of pattern and metacharacters is know 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 typeing 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 sensetive)

```
: 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 Home Next
Getting started with ex Up 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

Coping 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
```

Okay! I will stop.

```
:1,$ p
```

I love linux.

It is different from all other Os

My brother Vikrant also loves linux.

He currently lerarns linux.

Linux is cooool.

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 Home Next
Deleting lines Up 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 not 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 lerarns 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	Home	Next
Coping lines	Up	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 cooool.
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 cooool.

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 cooool.
:g/\<Linux\>/p
Linux is cooool.
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 cooool.

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 cooool.
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 cooool.
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 cooool.
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.

Using range of characters in regular expressions

Try the following command
:g/Linux/p
Linux is cooool.
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 cooool.
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 (1), 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 cooool.
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 . (DOT) or * loss 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.

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 cooool.
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) Note: 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

Prev Home Next
Converting lowercase character to uppercase Up Getting Starting with awk

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

'{ print $1 $2 "--> Rs." $3 * $4 } '	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 Home Next
awk Revisited Up 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)

Prev Home Next
Getting Starting with awk Up Doing arithmetic with awk

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 print 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 their 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 Home Next
Doing arithmetic with awk Up 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 try 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
%c	Character	{ isminor = "y" printf "%c" , isminor }
%d	Decimal number such as 10,-5 etc	{ n = 10 printf "%d",n }
%x	Hexadecimal number such as 0xA, 0xffff etc	{ n = 10 printf "%x",n }

%s	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 Home Next
User Defined variables in awk Up 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 equvalent to gtotal = gtotal + total. Here += is called assignment operator. You can use following assignment operator:

Assignment operator	Use for	Example	Equivalent to

+=	Assign the result of addition	a += 10
		d += c
			a = a + 10
			a = a + c

-=	Assign the result of subtraction	a -= 10
		d -= c
			a = a - 10
			a = a - c

*=	Assign the result of multiplication	a *= 10
		d *= c
			a = a * 10
			a = a * c

%=	Assign the result of modulo	a %= 10
		d %= c
			a = a % 10
			a = a % c

Prev Home Next
Use of printf statement Up if condition in awk

if condition in awk

General syntax of if condition is as follows:
Syntx:

```
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 define 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

```
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
}
```
	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.

}	Stop all inputted lines are process.

```
END {
  printf "\nGoodbuy %s\n" , ENVIRON["USER"]
}
```
	END patterns start here. Which says currently log on user Goodbuy.

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	Home	Next
Use of Format Specification Code	Up	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 the 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:

for(i=0;i<NF;i++)	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++)
printf "Welcome %s, %d times.\n" ,ENVIRON ["USER"], i	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 Home Next
if condition in awk Up 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 usguall it shoude installed under your-home-directory/bin (something like /home/vivek/bin)
ls -1 > /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.$$
else
  echo -e "\n$0: Fatal error - $awkspath not found"
  echo -e "\nMake sure \$awkspath is set correctly in $0 script\n"
fi
	See if rename.awk script installed, if not installed give error message on screen. If installed call the rename.awk script and give it /tep/tepdb.$$ 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
isdir1 = "[-d " $1 "] "	This expression is quite tricky. Its something as follows: isdir1 = [-d $1] 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.
isdir2 = "[-d " $2 "] "	As above except it test for second field as isdir2 = [-d $2] i.e. Whether second field is directory or not.

scriptname = "up2low"
awkscriptname = "rename.awk"
	Our shell script name (up2low) and awk script name (rename.awk).

sfile = $1	Source file
dfile = $2	Destination file

if (sfile == scriptname || sfile == awkscriptname)
  next
	Make sure we don't accidentally rename our own scripts, if scripts are in current working directory

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
}
	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.

`else if ( sfile == dfile )`
`{`
`  printf "Skiping, \"%s\" is alrady in lowercase\n",sfile`
`  next`
`}`
	If both source and destination file are same, it mean file already in lower case no need to rename it to lower case.

`else # everythink 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 Home Next

Loops in awk Up 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

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 Home Next
awk miscellaneous Up 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 Home Next
Redirecting the output of sed command Up 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 promote:

```
$ 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 "10000000". 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 explianed as follows:

Command	Explnation
^	Beginning of line
*	Find the asterisk or star (\ remove the special meaning of '*' metacharacter)
..	Followed by any two character (you can also use ** i.e. $ sed -n '/^***$/p' demofile2)
$	End of line (So that only three star or asterisk will be matched)

/p	Print the pattern.

Even you can use following expression for the same purpose
$ sed -n '/^*\{2,3\}$/p' demofile2

Now following command will find out lines between *** and *** and then delete all those line
$sed -e '/^*\{2,3\}$/,/^*\{2,3\}$/d' 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 > rmblksp
/^$/d
s/ */ /g<
```

Run above script as follows:

```
$ sed -f rmblksp demofile3
Welcome to word 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)Exapand MH state value to Maharastra
3)Find all blank line and replace with actual line (i.e. ========)
4)Instert 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/Maharastra/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 mkchgfrddb friends > updated_friendsdb
$ cat updated_friendsdb

Above script can be explained as follows:

Expression	Meaning
s/A.bad/Aurangabad/g	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.
s/MH/Maharastra/g	Substitute Maharastra for MH
s/^$/==========/g	Substitute blank line with actual line

/V.K. /{
N
N
a\\
email:vk@fackmail.co.in
}
	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 Home Next
How to write sed scripts? Up 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 the 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 supplies 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:

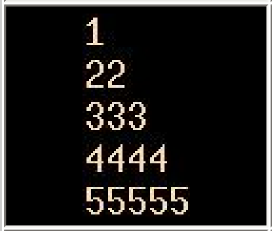
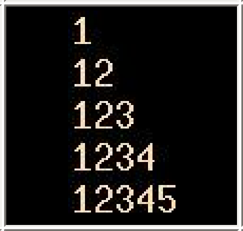
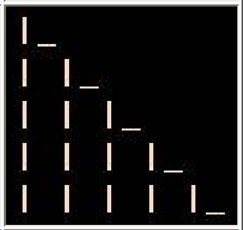
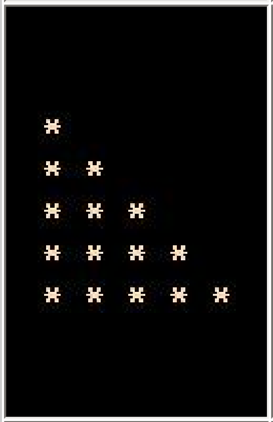
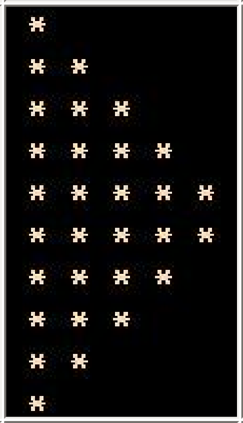
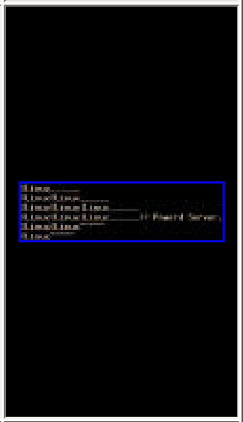
Menu-Item	Purpose	Action for Menu-Item
Date/time	To see current date time	Date and time must be shown using infobox of dialog utility
Calendar	To see current calendar	Calendar must be shown using infobox of dialog utility

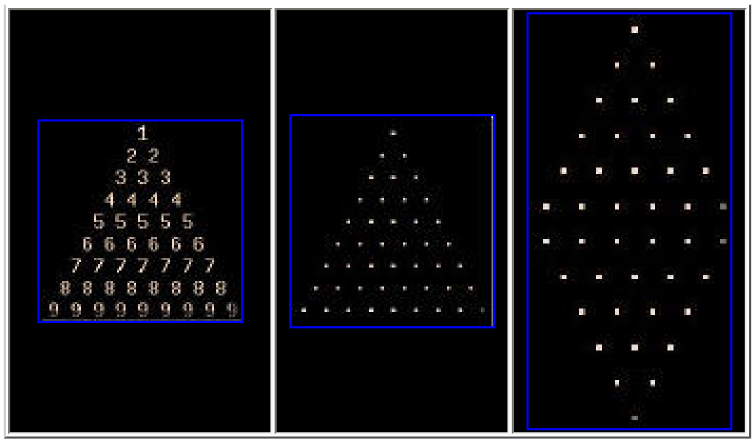
Delete	To delete selected file	First ask user name of directory where all files are present, if no name of directory given assumes current directory, then show all files only of that directory, Files must be shown on screen using menus of dialog utility, let the user select the file, then ask the confirmation 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
10) Show mouse settings
11) Show computer cpu information like processor type, speed etc
12) Show memory information
13) Show hard disk information like size of hard-disk, cache memory, model etc
14) File system (Mounted)
Answer: See Q19 shell Script.

Q.20.Write shell script using for loop to print the following patterns on screen

for2	for3	for4
 1 22 333 4444 55555	 1 12 123 1234 12345	 |_ | |_ | | |_ | | | |_ | | | | |_
for5	for6	for7
 * * * * * * * * * * * * * * *	 *	
for8	for8	for9



```
        1
       2 2
      3 3 3
     4 4 4 4
    5 5 5 5 5
   6 6 6 6 6 6
  7 7 7 7 7 7 7
 8 8 8 8 8 8 8 8
9 9 9 9 9 9 9 9 9
```

```
        .
       . .
      . . .
     . . . .
    . . . . .
   . . . . . .
  . . . . . . .
 . . . . . . . .
. . . . . . . . .
```

```
     .
    . .
   . . .
  . . . .
 . . . . .
. . . . . .
. . . . . .
 . . . . .
  . . . .
   . . .
    . .
     .
```

Answer: Click on above the links to see the scripts.

Q.21.Write shell script to convert file names from UPPERCASE to lowercase file names or vice versa.
Answer: See the rename.awk - awk script and up2sh shell script.

```
#!/bin/bash
#
# Shell Scripting Tutorial 1.05r3
#
# Written by Vivek G. Gite <vivek@nixcraft.com>
#
year=0

echo -n "Enter the year, I will tell you whether its Leap year or Not? "
read year

ans=$(($year % 4))  # or try ans=`expr $year % 2`

if [ $ans -eq 0 ]; then
   echo "$year is Leap Year"
else
   echo "$year is NOT Leap Year"
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/
#

#
# Declare the array of 5 subscripts to hold 5 numbers
#
declare nos[5]=(4 -1 2 66 10)

#
# Prints the number befor sorting
#
echo "Original Numbers in array:"
for (( i = 0; i <= 4; i++ ))
do
  echo ${nos[$i]}
done

#
# Now do the Sorting of numbers
#

for (( i = 0; i <= 4 ; i++ ))
do
   for (( j = $i; j <= 4; j++ ))
   do
      if [ ${nos[$i]} -gt ${nos[$j]}  ]; then
           t=${nos[$i]}
           nos[$i]=${nos[$j]}
           nos[$j]=$t
      fi
   done
done

#
# Print the sorted number
#
```

```
echo -e "\nSorted Numbers in Ascending Order:"
for (( i=0; i <= 4; i++ ))
do
  echo ${nos[$i]}
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/
#

avg=0
temp_total=0
number_of_args=$#

#
# First see the sufficent cmd args
#
if [ $# -lt 2 ] ; then
   echo -e "Opps! I need atleast 2 command line args\n"
   echo -e "Syntax: $0: number1 number2 ... numberN\n"
   echo -e "Example:$0 5 4\n\t$0 56 66 34"
   exit 1
fi

#
# now calculate the average of numbers given on command line as cmd args
#

for i in $*
do

   #  addition of all the numbers on cmd args
   temp_total=`expr $temp_total + $i`

done

avg=`expr $temp_total / $number_of_args`
echo "Average of all number is $avg"



#
# ./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/
#
n=0
hex=0
echo "Decimal to hexadecimal converter ver. b0.1"
echo -n "Enter number in decimal format : "
read n
hex=`echo "obase=16;ibase=10; $n" | bc`
echo "$n is equivalent \"$hex\" in hexadecimal"




#
# ./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/
#

n=0
on=0
fact=1

echo -n "Enter number to find factorial : "
read n

on=$n

while [ $n -ge  1 ]
do
  fact=`expr $fact \* $n`
  n=`expr $n - 1`
done

echo "Factorial for $on is $fact"

#
# ./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/
#
# Remark : Use this script to add new domain zone and www host useful for
#          vhosting
#
# AddDomain.sh ver. b0.3
#
#
# Change following varibles to point out correct entries
#
#
NAMED_CONF_PATH="/etc/named.conf"  # named.conf file location with path
BIND_DB_PATH="/var/named"          # do not append / at the end

SERIAL_NO_COUNT="/var/named/AddDomain" # Where to get serial number count

MAIL_SERVER="mail.ecyberciti.com." # MX Pointer i.e. All Mail forword to this
                                   # domain will goes to this mail server

NS1="ns1.ecyberciti.com."          # First Name Server
NS2="ns2.ecyberciti.com."          # Second Name Server


ADMIN_EMAIL="hostmaster.ecyberciti.com." # Hostmaster e-mail ID

# Input for - UDVs

ZONE_DB=""
domainname=""
ipadd=""
serial_no=1                # Default

#
# Main script begins here
#
echo -n "Enter Domain Name : "
```

```
read domainame

ZONE_DB="$BIND_DB_PATH/$domainame.zone"

echo -n "Enter IP Address for $domainame : "
read ipadd

#
# Find the serial_no count for domain
#

if ! [ -f $SERIAL_NO_COUNT ] ; then
   echo "Init... Serial Number count to default 1"
   echo "1" > $SERIAL_NO_COUNT
   serial_no=1
else # get the last time saved serial_no count
   serial_no=`cat $SERIAL_NO_COUNT`
   serial_no=`expr $serial_no + 1`
   echo "$serial_no" > $SERIAL_NO_COUNT
fi

#
# See if domain alrady exists or not
#

if  grep \"$domainame\" $NAMED_CONF_PATH > /dev/null  ; then
    echo "Domain $domainame already exists, please try another domain"
    exit 1
fi

#
# Make sure its valid IP Address
#
if which ipcalc > /dev/null ; then

 ipcalc -ms $ipadd > /dev/null
 if ! [ $? -eq 0 ] ; then
    echo -e "*** Bad ip address: $ipadd\nTry again with correct IP Address."
    exit 2
 fi
else
  echo "Warning can't validate the IP Address $ipadd"
fi
```

```
#
# Open the named.conf file and append the entries
#
echo "zone  \"$domainame\" {" >> $NAMED_CONF_PATH
echo "      type master;  " >> $NAMED_CONF_PATH
echo "      file \"$domainame.zone\";" >> $NAMED_CONF_PATH
echo "};" >> $NAMED_CONF_PATH

#
# Crate zone file for our domain
#
echo "$domainame.  IN  SOA  $NS1  $ADMIN_EMAIL  ("       > $ZONE_DB
echo "            $serial_no      ;serial"         >> $ZONE_DB
echo "            28800           ;refresh"  >> $ZONE_DB
echo "            7200            ;retry"     >> $ZONE_DB
echo "            604800          ;expire"    >> $ZONE_DB
echo "            86400           ;TTL">> $ZONE_DB
echo "                               )"      >> $ZONE_DB
echo ";"                                    >> $ZONE_DB
echo ";Name Servers"                                >> $ZONE_DB
echo "$domainame.             IN       NS       $NS1" >> $ZONE_DB
#
# See if we want Name server 2
#
if [ "$NS2" != "" ]; then
 echo "$domainame.             IN       NS       $NS2" >> $ZONE_DB
fi
echo ";"                                    >> $ZONE_DB
echo ";Mail Servers"                                >> $ZONE_DB
echo "$domainame.      IN      MX       10       $MAIL_SERVER" >> $ZONE_DB
echo ";"                                    >> $ZONE_DB
echo ";IP addresses $domainame "                       >> $ZONE_DB
echo "$domainame.      IN      A       $ipadd" >> $ZONE_DB
echo "www                  IN      A       $ipadd" >> $ZONE_DB

echo "$domainame ($ipadd) Addedd successfully."




#
# ./ch.sh: vivek-tech.com to nixcraft.com referance converted using this tool
# See the tool at http://www.nixcraft.com/uniqlinuxfeatures/tools/
```

```
#
```

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 used as Quick Linux Command Reference guide.

More information on upcoming edition of this tutorial.

Appendix - A

Linux File Server Tutorial (LFST) version b0.1 Rev 3

This is the first PART-I of Linux File Server Tutorial (LFST b0.1) which is distributed with LSST v.1.05, Aug.-2001. Next PART-II (LFST b0.2) will have more technical details about Linux file system and more utilities are cover in depth.
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.
Latest version of this can be obtained from: http://www.nixcraft.com/docs/
This tutorial uses the Red Hat Linux version 6.2. and Red Hat Linux 7.2.

- What is a File?
- What directory is?
- How to identify directory and file in Linux?
- How to get more information about file in Linux?
- What is Path?
- What is mount and umount command and why they are required?
    - How to Use Floppy disk (How to mount floppy disk)?
    - How to Use CD-ROM disk (How to mount CD-ROM disk)?
    - How to Use C: under Linux (How to mount DOS partition under Linux disk)?
    - How to Unmount device?
    - How to unmount floppy disk?
    - How to unmount CD-ROM disk?
    - How to unmount DOS partition (C:)?
    - How to Copy file?
        - (a) From Floppy
        - (b) From CDROM
        - (c) From MS-DOS/Windows Partition
        - (d) From One directory to another
        - (e) How do I copy all files and subdirectory and files?
        - (f) How do I preserve the all files ownership permission, time stamp while coping?
    - Delete file
        - (a) Normal file delete
        - (b) How do I delete all files and subdirs from directory
    - Rename file
        - (a) Normal rename file

        - (b) How do I rename all my .htm files to .html files?
        - (c) How do I rename my all files like abczz01.htm, abczz02.htm.., to abcyy01.htm, abcyy02.htm
    - Move file
    - Changing file date & time stamp
    - Creating symbolic link
    - Changing file permission or ownership
- Searching files
    - (a) How do I find all file have *.c ?
    - (b) How do I find all core file and delete them?
    - (c) How do I find all *.c files having 2kb file size?
    - (d) How do I find all files access 2 days before?
    - (e) How do I find all files NOT accessed in a given period (2 days)?
    - (f) How do I find all special block files in my system?
    - Using locate command
    - Using whereis
        - (a) How do I find location of ls (binary program) and it's man page location?
- Compressing / Decompressing files Using tar

What is a File?
File are collection of data items stored on disk. Or it's device which can store the information/data/music/picture/movie/sound/book; In fact what ever you store in computer it must be inform of file. Files are always associated with devices like hard disk ,floppy disk etc. File is the last object in your file system tree.

In Linux files have different types like

- Executable files   (stored in /bin, /usr/bin)
- Special files (stored in /dev)
- Configuration files (stored in /etc)
- Directory
- User files or ordinary files etc

And the process which manage the Files i.e. creation, deletion, copying of files and giving read, write access of files to user is called File management.

Normally you can perform following operation on files

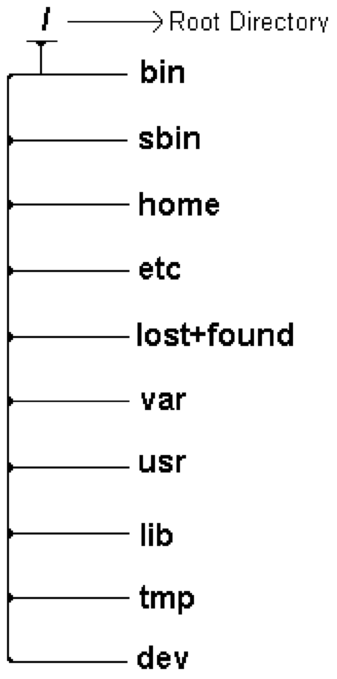
- Copy file
- Delete file
- Rename file
- Move file
- Changing file date & time stamp
- Creating symbolic link
- Changing file permission or ownership
- Searching files

- Compressing / Decompressing files
- Comparison between files
- Printing files on printer
- Sorting files etc

What directory is?
Directory is group of files. Directory is divided into two types as

(1) Root directory : Strictly speaking, there is only one root directory in your system, which is shown by / (forward slash), which is root of your entire file system. And can not be renamed or deleted.
(2) Sub directory : Directory under root (/) directory is subdirectory which can be created, rename by the user. For e.g. bin is subdirectory which is under / (root) directory.
Directories are used to organize your data files, programs more efficiently.

When you install Linux (Red Hat ver. 6.x), by default following directories are created by Linux



Directory Name	Use to store this kind of files
/bin	All your essential program (executable files) which can be use by most of the user are store here. For e.g. vi, ls program are store here.
/sbin	All super user executable (binaries) files are store which is mostly used by root user . As mentioned in Red Hat Linux documentation: "*The executables in /sbin are only used to boot and mount /usr and perform system recovery operations.*"
/home	Your sweet home. Mostly you work here. All the user have their own subdirectory under this directory.
/etc	Configuration file of your Linux system are here. For e.g. smb.conf - Samba configuration file. It may contained sub directories like xinetd.d or X11 which contains more configuration files related to particular application or program. See the rest of subdirectories.

/lost+found	When your system crashes, this is the place where you will get your files. For e.g. You have shutdown your computer without unmounting the file system. Next time when your computer starts all your 'fsck' program try to recover you file system, at that time some files may stored here and may be removed from original location so that you can still find those file here.
/var	Mostly the file(s) stored in this directory changes their size i.e. variable data files. Data files including spool directories and files, administrative and logging data, and transient and temporary files etc.
/usr	Central location for all your application program, x-windows, man pages, documents etc are here. Programs meant to be used and shared by all of the users on the system. Mostly you need to export this directory using NFS in read-only mode. Some of the important sub-directories are as follows: bin - sub-directory contains executables sbin - sub-directory contains files for system administration i.e. binaries include - sub-contains C header files share - sub-contains contains files that aren't architecture-specific like documents, wallpaper etc. X11R6 - sub-contains X Window System
/lib	All shared library files are stored here. This libraries are needed to execute the executable files (binary) files in /bin or /sbin.
/tmp	Temporary file location
/dev	Special Device files are store here. For e.g. Keyboard, mouse, console, hard-disk, cdrom etc device files are here.
/mnt	Floppy disk, CD-Rom disk, Other MS-DOS/Windows partition mounted in this location (mount points) i.e. temporarily mounted file systems are here.
/boot	Linux Kernel directory
/opt	Use to store large software applications like star office.
/proc	This directory contains special files which interact with kernel. You can gathered most of the system and kernel related information from this directory. To clear your idea try the following commands: # cat cpuinfo # cat meminf # cat mounts See the System information shell script for more information.

How to identify directory and files in Linux?
Use any of the following two ways:
(1) When you give command like ls -l you will see something like

```
drwxr-x--x 1   root  root   62 Mar 11  21:27   one.c
```

Here the first string (word) i.e. drwxr-x--x will help you to identify whether its directory, file or any other special file.

```
drwxr-x--x
0123456789
```

0 - Can be any one of the following

b block device hard disk, floppy disk

c character device sound, midi, com port

d directory (If its d it means its directory - 1st method)

p name pipe (FIFO)

f regular file

l symbolic link s socket

123 shows owner rights

456 shows group rights

789 shows others rights

Where

r : read permission

w : write permission

x : execute permission

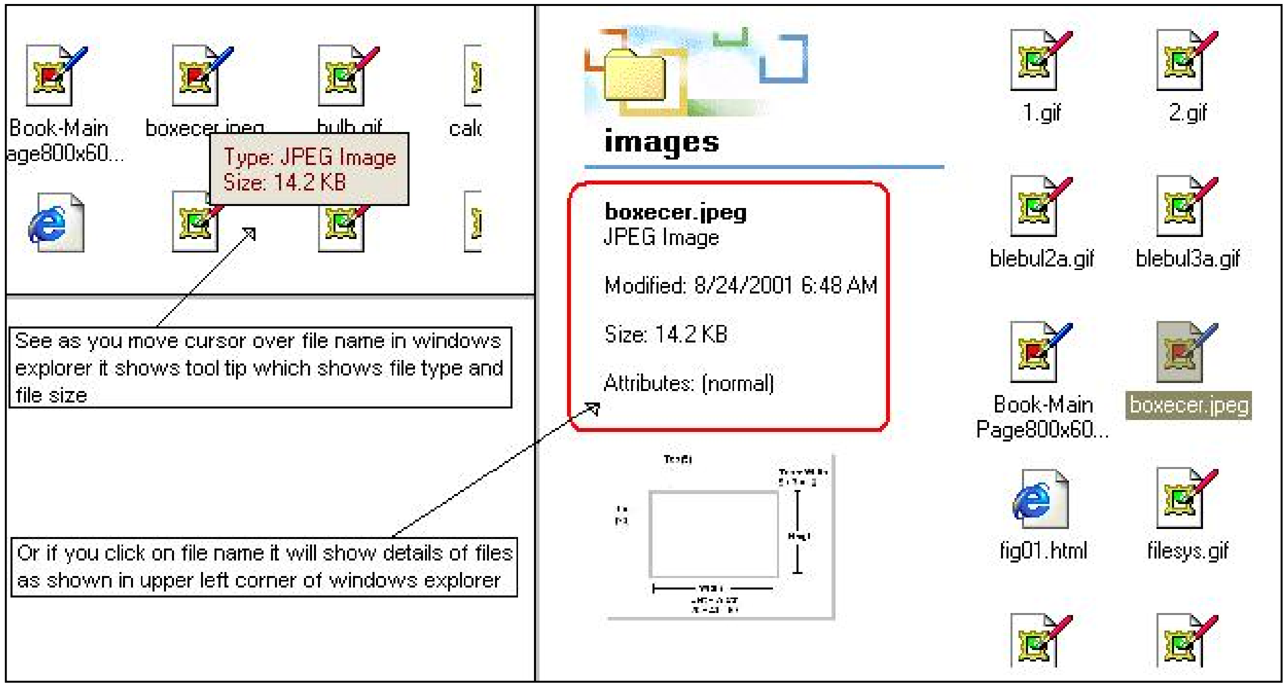
- : No permission set for this bit.

If 0 field is - or f or l it means its file.

(2) You can type ls -p command and if at the end of directory / is appended then its directory.(2nd method). Or try the script dir as shown in examples chapter.

How to get more information about file in Linux?

In MS-Windows as you move mouse pointer/curser over file name (in windows explorer) it will show the tool tip in which you will get the information about file.



Type: JPEG Image
Size: 14.2 KB

See as you move cursor over file name in windows explorer it shows tool tip which shows file type and file size

Or if you click on file name it will show details of files as shown in upper left corner of windows explorer

boxecer.jpeg
JPEG Image

Modified: 8/24/2001 6:48 AM

Size: 14.2 KB

Attributes: (normal)

Getting file information from shell prompt:

You can try any of the two method to get more file information:

(1) Using stat command: Print file information.

Syntax: stat {file-name}

Examples:

```
$ stat one.c
```

```
$ stat /dev/hda2
```

On my PC it gave output as follows

```
File: "/dev/hda1"
  Size: 0          Filetype: Block Device
  Mode: (0660/brw-rw----)       Uid: (    0/    root)  Gid: (    6/    disk)
Device:  3,2   Inode: 176841    Links: 1       Device type:  3,1
Access: Wed May  6 02:02:26 1998(01040.19:40:07)
Modify: Wed May  6 02:02:26 1998(01040.19:40:07)
Change: Mon Mar  5 07:50:55 2001(00006.13:51:38)
```

(2) Using file command: Print magic information or file type on screen.

Syntax: file   {file-name}

Examples:

```
$ file  foo
```

```
foo: English text
```

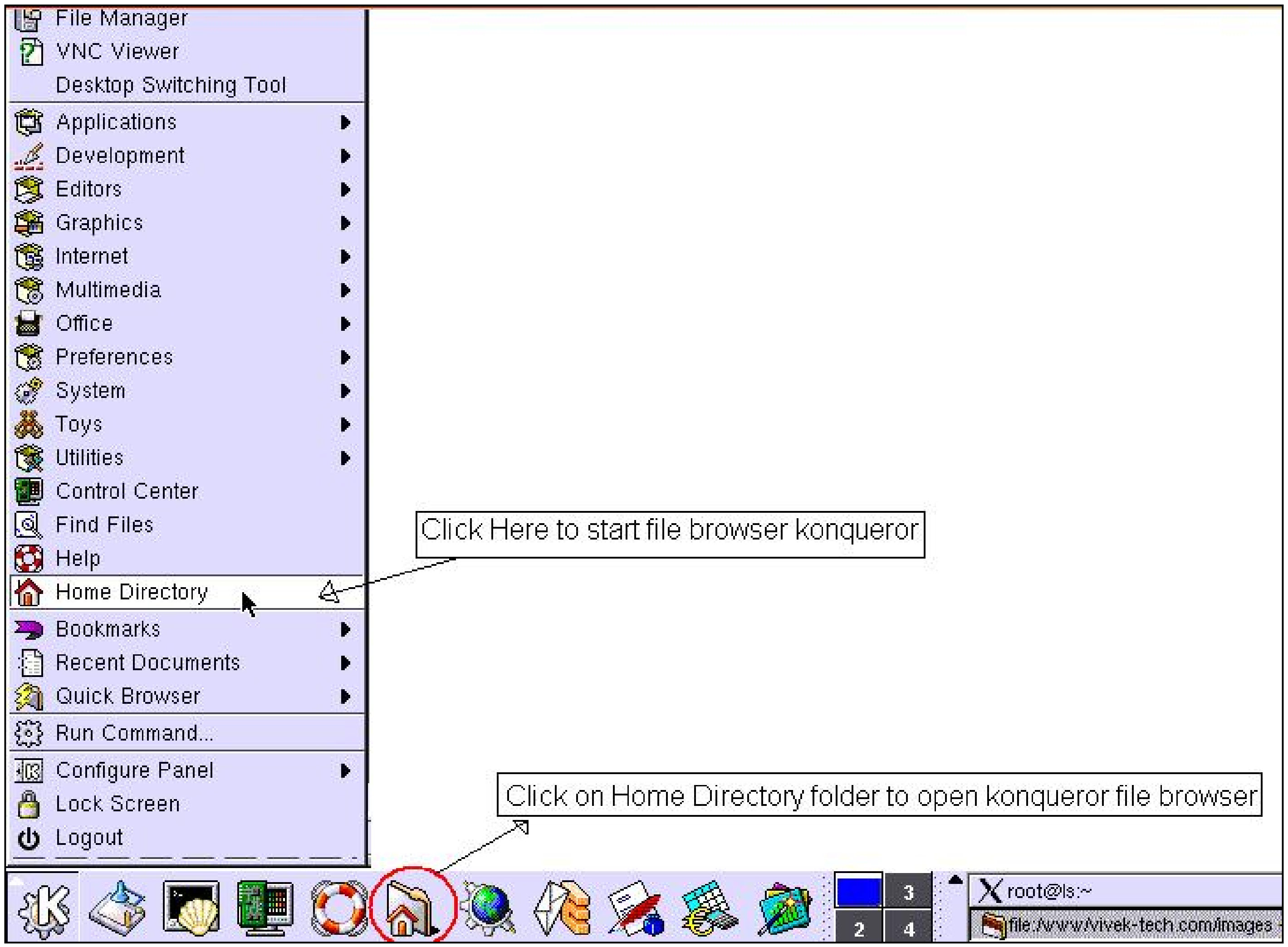
```
$ file -s /dev/hda{1,2,3,4,5,6,7,8}
```

```
/dev/hda1: x86 boot sector, system MSWIN4.1, FAT (16 bit)
/dev/hda2: Linux/i386 ext2 filesystem
/dev/hda3: x86 boot sector, extended partition table
/dev/hda4: x86 boot sector, extended partition table
/dev/hda5: x86 boot sector, FAT (32 bit)
/dev/hda6: x86 boot sector, FAT (32 bit)
/dev/hda7: x86 boot sector, FAT (32 bit)
/dev/hda8: empty
```

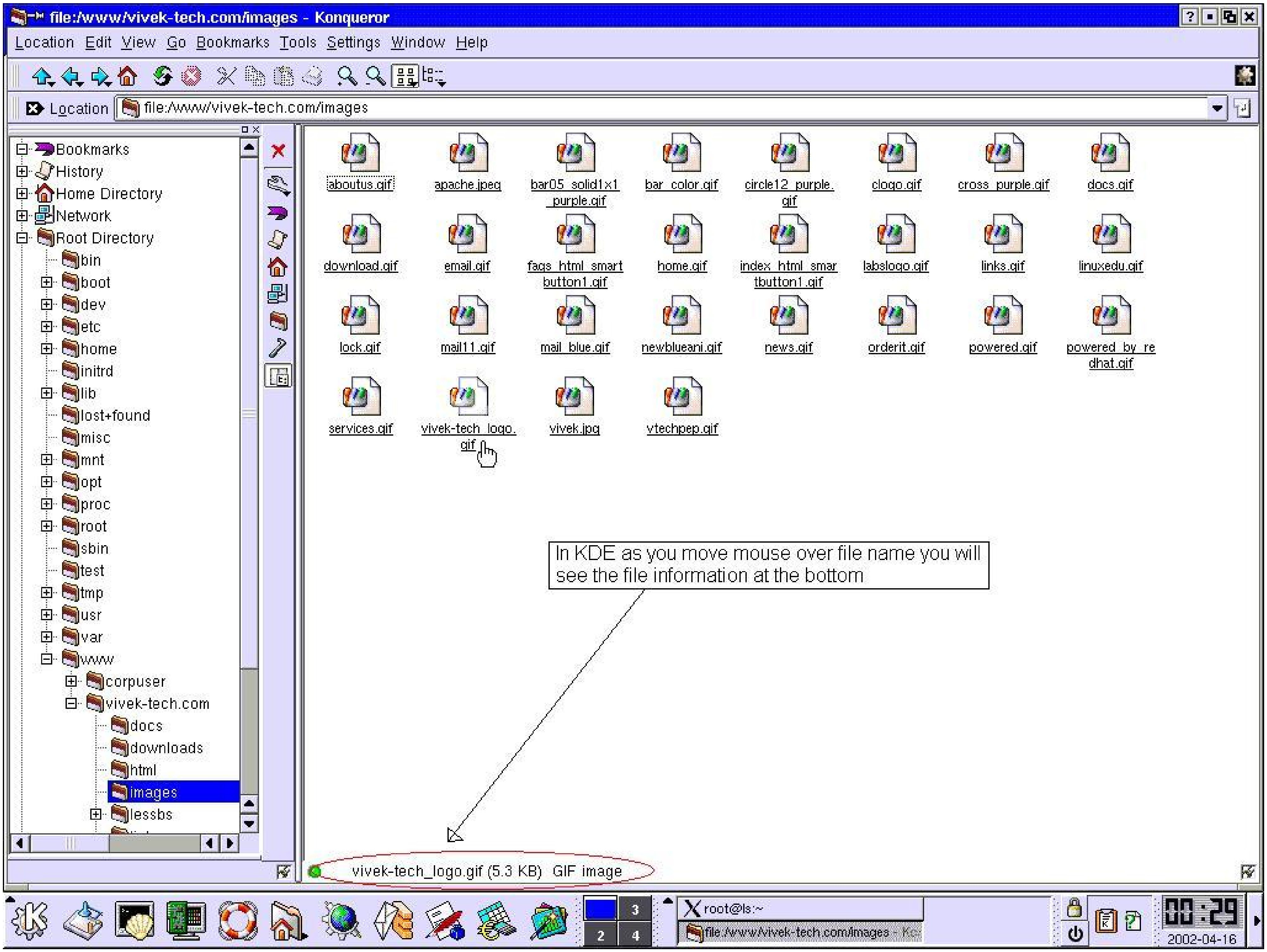
Note that for first example file type for foo is "English Text" and for device file its as above.

Getting file information from KDE's File Manager:

If you have X - windows installed then open the KDE and kdes file browser konqueror by clicking on Home Directory folder on Desktop or from Start Application (K) > Home Directory: (As shown in following figure)



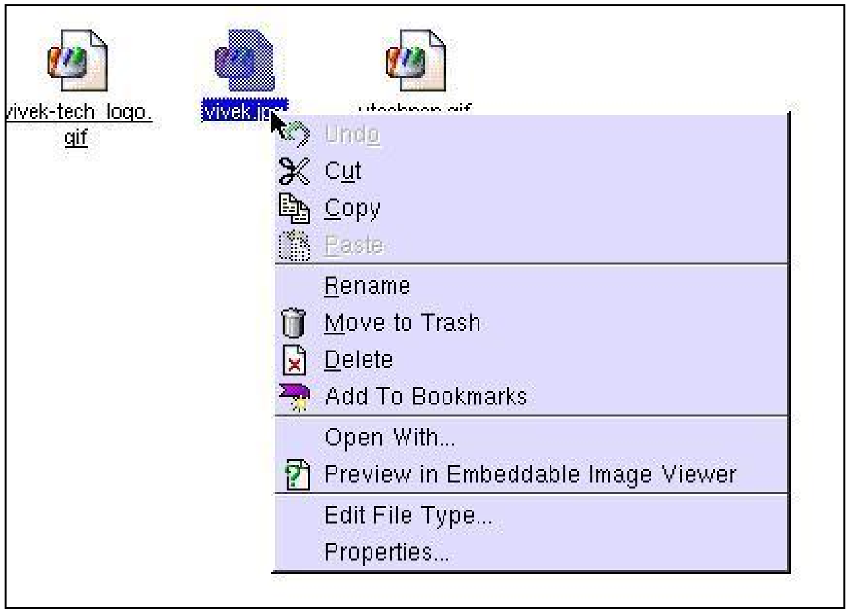
Once konqueror open up, you can use it just like windows explorer . As you move mouse pointer/curser on any file it will show you file type and size at the bottom of the screen as shown in following figure:



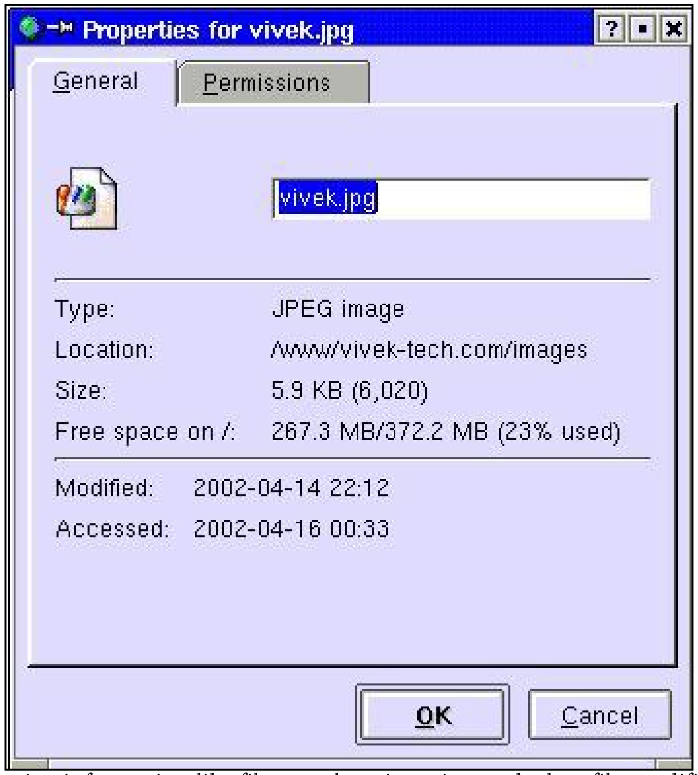
In KDE as you move mouse over file name you will see the file information at the bottom

vivek-tech_logo.gif (5.3 KB) GIF image

Even you can right click on file name as shown in following figure:



Once you click on Properties option you will get the properties windows as shown below with all file information you need.



vivek.jpg

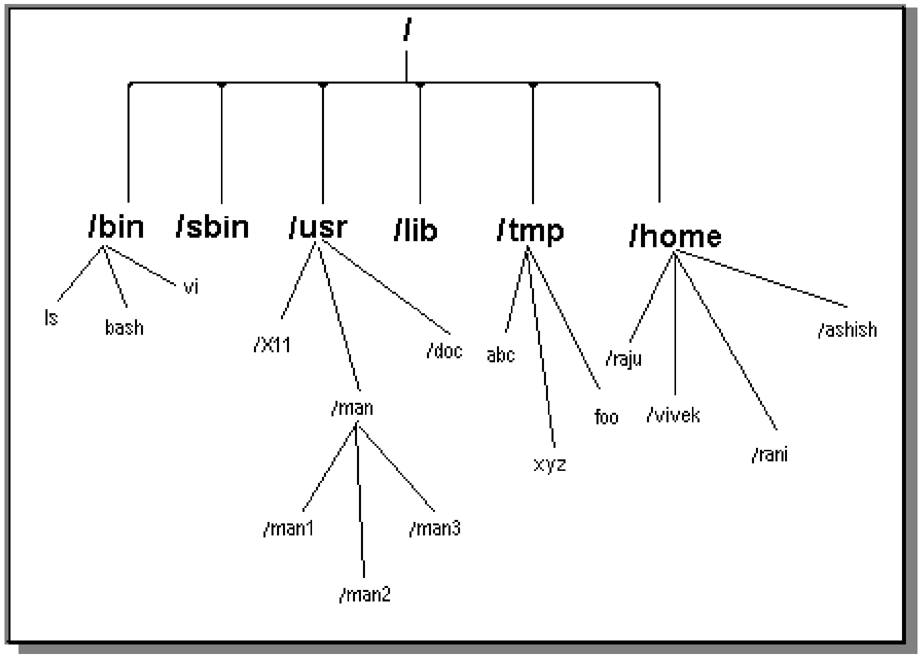
Type: JPEG image
Location: /www/vivek-tech.com/images
Size: 5.9 KB (6,020)
Free space on /: 267.3 MB/372.2 MB (23% used)

Modified: 2002-04-14 22:12
Accessed: 2002-04-16 00:33

File Properties showing information like file type, location, size, and when file modified or accessed etc

What is Path?

A path is route you use to go from one directory to another directory or its way to tell Linux which file you want from tree type of file system.



NOTE for above tree: bin, sbin, usr, lib, tmp, home, x11, man, man1, man2, man3, doc, raju, vivek, rani, ashish are all directories and all other are files.

Now consider above file system tree, Here what is the path of abc file ? First locate abc file, Its under tmp directory and tmp under root (/) directory so path for abc file is as follows

/tmp/abc

Try to write down path for following

1) ls command

2) X11 directory

3) man1

4) xyz file

5) root directory

Now suppose you want to copy abc file to you home directory (lets say /home/vivek) then command will be

```
cp  /tmp/abc   /home/vivek
```

Path is separated by / (forward slash).

Now write command for following (I assume that you are in /home/rani directory)

1) Copy the file foo to /home/vivek directory

2) Copy the all files of /bin directory to /tmp directory

3) Delete all files from /tmp directory

4) Copy /bin/ls to your home directory

5) Copy all files from man3 directory to /tmp directory

6) Copy all files whose first name beginning with a from man3 directory to /tmp/fun directory

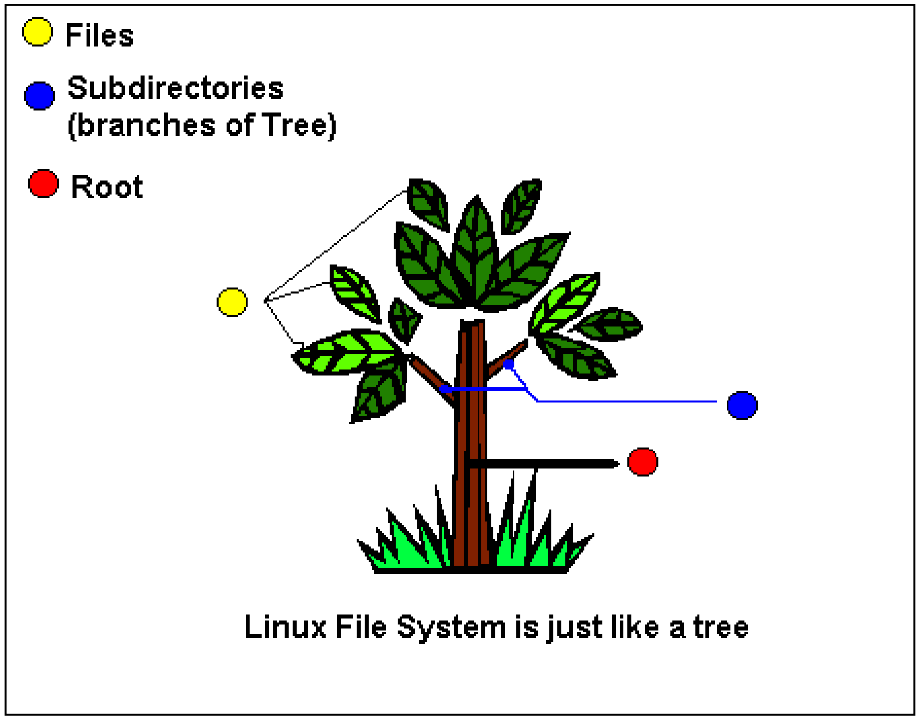
What is mount and umount command and why they are required?

In MS-DOS, you see A: or B: or C: or D:, what is that? It's called drive, which is nothing but group of all directories and the drive name is given according to disk type by MS-DOS Os. For e. g. First floppy drive on IBM PC is A: (Read as A drive), Second is B: etc. When you want to use floppy you insert floppy disk into floppy drive and simply change drive later to A: as follows

```
C:\> A:

A:\> DIR
```

This is the feature of MS-DOS/Windows 9x Os. But how you will use floppy under linux? Linux don't understand drive letter as used in MS-DOS.

Let's try to understand Linux's tree file system


Files
Subdirectories (branches of Tree)
Root
Linux File System is just like a tree

From above Figure you can see there is only one root (/ directory) for tree, this root of tree divided into several branches (Subdirectories) and branches have different flowers, fruits, leaves etc (Files).

Under Linux all files, directory, cd-rom, floppy drive are all based on tree structure with root at the top. In fact many of the directories in the tree are physical devices like cd rom , floppy disk, OR partition of hard-disk.(may be partition of different remote computer). When such disk (floppy,cdrom etc) OR partition is attached to the file tree at a directory know as mount point and all files below that are referred as file system.

/dev directory under Linux contains special files which are require to access hardware devices such as floppy disk, CD-ROM disk, Another partition, tape drive etc.

For e.g.

IDE Devices (CD-ROM/RW or Hard Disk etc)

/dev/hdx -- Where x is the first IDE physical hard disk connected to your system. X can be a,b,c,d etc. hd means hard disk.

Where

/dev/hda means first IDE physical hard disk.

/dev/hdb means second IDE physical hard disk or cdrom drive.

Here /dev/hda1 refers to first partition of your disk i.e. C: in DOS.

SCSI Devices (CD-Rom/RW or Hard disk etc)

/dev/sdx -- Where x is the first SCSI physical hard disk connected to your system. X can be a,b,c,d etc. sd means SCSI disk.

Where

/dev/sda means first SCSI physical hard disk.

/dev/sdb means second SCSI physical hard disk or cdrom drive.

Here /dev/sd1 refers to first partition of your disk i.e. C:.

Floppy Disk

/dev/fdx -- Where x can be 0 or 1. Your first floppy disk drive (A:) is know as /dev/fd0, second floppy disk drive is know as /dev/fd1.

In MS-DOS/Win 9x/NT	Device	In Linux
C:	Hard disk	/dev/hda1 (IDE)
A:	Floppy disk	/dev/fd0
D:	CD-Rom	/dev/hdb (IDE)
C:	Hard disk	/dev/sda1 (SCSI)

But how to use them?

Make sure mount point exists, by default in Red Hat Linux v 6.x and 7.x, it create following mount points

Directory name	Device
/mnt/cdrom	CD-Rom
/mnt/floppy	Floppy Disk

You can create your own mount point if above not exist simply by typing following command

```
$ su -l
passowrd
# mkdir /mnt/cdrom
# mkdir /mnt/floppy
```

Command	Use
$ su -l	Become root user, otherwise mounting is not possible
# mkdir /mnt/cdrom # mkdir /mnt/floppy	Create mount point directory

Also you can create mount point for your MS-DOS/Windows partition as follows (I assume that you have install DOS/Windows in C: i.e. /dev/hda1 (i.e. for IDE or for SCSI use /dev/sda1))

```
$ su -l
# mkdir /mnt/c
```

Now to mount device (CDROM,FLOPPY DISK etc) use following mount command

Syntax: mount   option {device} {mount-point}

Option:

-t   type   To specify file system being mounted.

For MS-DOS use msdos

For Windows 9x use vfat (Long file name support under windows 95/NT etc)

iso9660 type is default and used by cd-roms.

-v   Its called Verbose mode which gives more information when you mount the file system.

How to Use Floppy disk (How to mount floppy disk)?

```
$ su -l
# mount /mnt/floppy
# cd /mnt/floppy
# ls
```

How to Use CD-ROM disk (How to mount CD-ROM disk)?

```
$ su -l
# mount /mnt/cdrom
# cd /mnt/cdrom
# ls
```

How to Use C: under Linux (How to mount DOS/Windows 9x (FAT 32) partition under linux disk)?

Use following commands for MS-DOS FAT 16 Partition:

```
$ su -l
# mount -t msdos /dev/hda1  /mnt/c
# cd /mnt/c
# ls
```

Use following commands for MS-Windows FAT 32 Partition:

```
$ su -l
# mount -t vfat /dev/hda1  /mnt/c
# cd /mnt/c
# ls
```

Process of attaching device to directory is called mounting and process which removes attached device from directory is called unmounting.

How to Unmount device?

Syntax: umount { device }
Syntax: umount -t type { device }

How to unmount floppy disk?

```
$ su -l
back to home dir, you can't unmount while you are in /mnt/floppy
# cd
# umount /mnt/floppy
```

How to unmount CD-ROM disk?

```
$ su -l
# cd
# umount /mnt/cdrom
```

How to unmount DOS partition (C:)?

```
$ su -l
# cd
# umount -t msdos /dev/hda1
```

How to Copy file?

(a) From Floppy

First Mount the floppy

```
$ su -l
# mount /mnt/floppy
# cd /mnt/floppy
# ls
```

Copy foo file from floppy disk to /home/vivek directory

```
# cp foo /home/vivek
```
Back to our home directory
```
# cd
```
Unmount the floppy disk
```
# umount /mnt/floppy
# logout
```

(b) From CDROM

Mount the CDROM as
```
$ su -l
# mount /mnt/cdrom
# cd /mnt/cdrom
# ls
```
Copy bar file to /home/vivek from CD-Rom
```
# cp bar /home/vivek
# cd
# umount /mnt/cdrom
# logout
```

(c) From MS-DOS/Windows Partition

Mount the partition as (Assumes MS-DOS/Windows 9x partition the first partition i.e. C:)
```
$ su -l
# mount -t msdos /dev/hda1 /mnt/c
# cd /mnt/c
# ls
```
Copy new.gif file from c: under Linux
```
# cp new.gif /home/vivek
# cd
# umount -t msdos /dev/hda1
# logout
```

(d) From One directory to another

Copy all C program files from /usr/pub directory to /home/vivek/AllCprog directory
```
$ cp /usr/pub/*.c /home/vivek/AllCprog
```

(e) How do I copy all files and subdirectory and files ?

Copy all directories, subdirectories and files from /usr/pub directory to /home/vivek
```
$ cp -R /usr/pub /home/vivek
```

(f) How do I preserve the all files ownership permission, time stamp while coping?
```
$ cp -P /usr/pub/* /home/vivek
```

Note:Try -a option if you want to preserve file attributes only.

Delete file

(a) Normal file delete
Syntax: rm {filename}
Example:$ rm foo

(b) How do I delete all files and subdirs from directory
Syntax: rm -rf {dirname}
Example:$ rm -rf uwfiles

Rename file

(a) Normal rename file
Syntax: mv {old-file-name} {new-file-name}
Example:$ mv foo bar

(b) How do I rename all my .htm files to .html files?
Opps! mv will not work, it works with two files only. Try *rename* as
Syntax: rename from to files-to-rename
Example:$ rename .htm .html *.htm

(c) How do I rename my all files like
abczz01.htm
abczz02.htm
abczz03.htm
abczz04.htm

to
abcyy01.htm
abcyy02.htm
abcyy03.htm
abcyy04.htm

Example:$ rename abczz abcyy *.htm

Move file

(a) Normal move
Syntax: mv {source} {destination}
Example: $ mv /home/vivek /mnt/floppy

Changing file date & time stamp

Syntax: touch {filename}
Example:
See the time

```
$ls -l foo
-rw-r--r-- 1 root root 284 Mar 11 21:36 foo
$touch foo
```
Now see the time after touch command
```
$ls -l foo
-rw-r--r-- 1 root root 284 Mar 21 22:37 foo
```

Creating symbolic link

Syntax: ln -s {file-name} {symbolic-link-name}
Example:
Our normal file
```
$ ls -l foo
-rw-r--r-- 1 root root 284 Mar 11 22:37 foo
```
Now create link to foo file, so whenever I refer to bar file I am referring to foo file
```
$  ln -s foo bar
$ ls -l bar
lrwxrwxrwx 1 root root 7 Mar 11 22:40 bar -> foo
```

Searching files

Find Command
Find can find files according to file name,size,creation time etc.
Syntax:
find {dir-name}   -name files-to-search   -print

(a) How do I find all file have *.c ?
```
$ find  / -name  *.c -print
```

(b) How do I find all core file and delete them ?
```
$ find /  -name core*  -ok rm {} \;
```

(c) How do I find all *.c files having 2kb file size?
Syntax: find -size number[ckwb]
number - It's 2 byte or 1 kilobyte etc
c - bytes
k - kilobytes
w - words (2-byte)
b - 512-byte block
Example:
```
$ find /  -name *.c -size 2k -print
```

(d) How do I find all files access 2 days before?
Syntax: find -type f -atime -days
Example:
```
$ find /home/vivek -type f -atime -2
```

(e) How do I find all files NOT accessed in a given period (2 days)?
Syntax: find -type f -atime +days
Example:
```
$ find /home/vivek -type f -atime +2
```

(f) How do I find all special block files in my system?
Syntax: find -type [bcdpfls]
b block device
c character device
d directory
p name pipe (FIFO)
f regular file
l symbolic link
s socket
Example:
```
$ find / -type b -print
```

Using whereis

Use to find binary, source and man pages files for command.
Syntax: whereis {filename}

(a) How do I find location of ls (binary program) and it's man page location?
```
$ whereis ls
```

Note that most of the above command can be done easily if X-Windows (KDE) is running; otherwise you can use above commands. I recommend to use above command (at least you should know how to use those commands) because on most of the servers, especially web servers, FTP, and Getaway boxes X windows is not installed due to performance and security issue. So you must know how to handles files (File Management) from shell prompt.

Prev Home Next
Up Appendix -B

About Author

Vivek G. Gite runs small firm called "Cyberciti Computer" and *nix Solution firm nixCraft Technologies. He his freelance software developer and also teaches computer hardware, networking and Linux/Unix to beginners. He is also working varioues 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.
or snail-mail:
Vivek G. Gite
D-401, Bharti Vihar,
Near PICT College, Dhankwadi,
Katraj, Pune - 411046.
Maharastra, INDIA.

Where do I find the latest version?
Please visit http://www.nixcraft.com/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.