**What Linux is?**

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

First, It’s available free of cost (You don't have to pay to use this OS, other OS like MS-Windows or Commercial version of UNIX may cost you money)

Second free means freedom to use Linux, i.e. when you get Linux you will also get source code of Linux, so you can modify OS according to your taste.

It also offers many Free Software applications, programming languages, and development tools etc. Most of the Program/Software/OS are under GNU General Public License (GPL).

Linux is developed under the GNU Public License. This is sometimes referred to as a "copyleft", to distinguish it from a copyright.

Under GPL the source code is available to anyone who wants it, and can be freely modified, developed, and so forth. There are only a few restrictions on the use of the code. If you make changes to the programs, you have to make those changes available to everyone. This basically means you can't take the Linux source code, make a few changes, and then sell your modified version without making the source code available.

**Who developed the Linux?**

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

**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 are available. Following are important Linux distributions.

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

## What is a File?

Files 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 in form 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 processes 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

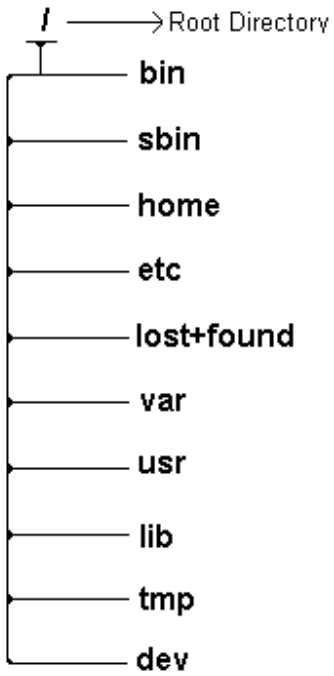
- Copy file
- Delete file
- Rename file
- Move file
- Changing file date & time stamp
- Creating symbolic link
- Changing file permission or ownership
- Searching files
- Compressing / Decompressing files
- Comparison between files
- Printing files on printer
- Sorting files etc

## 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, by default following directories are created by Linux



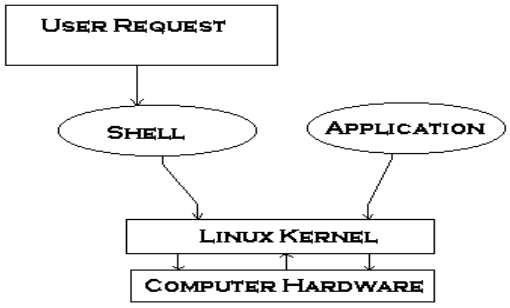
| Directory Name | Use to store this kind of files |
|---|---|
| /bin | Your essential programs (executable files) which can be used by most of the user are stored here. For e.g. vi, ls program are stored here. |
| /sbin | All super user executable (binaries) files are stored here which is mostly used by root user. As mentioned in 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. Every user has their own subdirectory under this directory. |
| /etc | Configuration file of your Linux system are stored here. For e.g. smb.conf - Samba configuration file. |
| /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:<br>**bin** - sub-directory contains executables<br>**sbin** - sub-directory contains files for system administration i.e. binaries<br>**include** - sub-contains C header files<br>**share** - sub-contains contains files that aren't architecture-specific like documents, wallpaper etc. |
| /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 stored 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 gather most of the system and kernel related information from this directory. To clear your idea try the following commands:<br>**# cat cpuinfo**<br>**# cat meminf** **# cat mounts** |

| | |
|---|---|

**What Kernel Is?**

Kernel is heart of Linux OS.

It manages resource of Linux OS. A resource 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 performs following task :-

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

**What is Linux Shell?**

Computer understands 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 a command language interpreter that executes commands read from the standard input device (keyboard) or from a file.

Several shells 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.<br>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 different command syntax and provides different built-in functions.

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

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

**What is Shell Script?**

Normally shells are interactive. It means shell accept command from you (via keyboard) and execute them. But if you use command one by one (sequence of 'n' number of commands), then 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 known as ***shell script***.

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

**Why to Write Shell Script?**

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

**Getting started with Shell Programming**

In this part of tutorial you are introduce to shell programming, how to write script, execute them etc. We will get 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
- Basic command navigation

**How to write shell script**

Following steps are required to write shell script:

(1) Use any editor like vi to write shell script with an extension .sh

(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.

```
$ vi first.sh
#
# 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 |
| #<br># My first shell script<br># | # followed by any text is considered as comment. Comment gives more information about script, logical explanation about shell script.<br>*Syntax:*<br># 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<br>*syntax:*<br>echo "Message" |

***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 : ";date
echo "Number of user login : " ; who | wc -l
echo "Calendar"
cal
exit 0
```

**Future Point:** At the end why exit 0 statement is used?

**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 (It's 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 case letters.

You can see system variables by giving command like **$env**, 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?

Ans.: (a) command is correct, since we have to print the contents of variable (HOME) and not the HOME. You must use **$** followed by variable name to print variables contains

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

**How to define User defined variables (UDV)**

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

'**value**' is assigned to given '**variable name**' and Value must be on right side = sign.

*Example:*

```
$ no=10# this is ok
$ 10=no# Error, NOT Ok, Value must be on right side of = sign.
```

To define variable called 'vech' having value Bus

```
$ vech=Bus
```

To define variable called n having value 10

```
$ n=10
```

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

(1) Variable name must begin with Alphanumeric character or underscore character (_), followed by one or more Alphanumeric character. For e.g. valid shell variable are as follows

**HOME**
**SYSTEM_VERSION**
**vech**
**no**

(2) Don't put spaces on either side of the equal sign when assigning value to variable. For e.g. In following variable declaration there will be no error

```
$ no=10
```

But there will be problem for any of the following variable declaration:

```
$ no =10
$ no= 10
$ no = 10
```

(3) Variables are case-sensitive, just like filename in Linux. For e.g.

```
$ no=10
$ No=11
$ NO=20
$ nO=2
```

Above all are different variable name, so to print value 20 we have to use $ echo $NO and not any of the following

`$ echo $no` # will print 10 but not 20

`$ echo $No`# will print 11 but not 20

`$ echo $nO`# will print 2 but not 20

(4) You can define NULL variable as follows (NULL variable is variable which has no value at the time of definition) For e.g.

$ vech=
$ vech=""

Try to print it's value by issuing following command

```
$ echo $vech
```

Nothing will be shown because variable has no value i.e. NULL variable.

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

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

To print or access UDV use following syntax

*Syntax:*

$variablename

Define variable vech and n as follows:

```
$ vech=Bus
$ n=10
```

To print contains of variable 'vech' type

```
$ echo $vech
```

It will print 'Bus', To print contents 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"
```

Q.1.How to Define variable x with value 10 and print it on screen.

**$ x=10**
**$ echo $x**

Q.2.How to Define variable xn with value Rani and print it on screen

**$ xn=Rani**
**$ echo $xn**

Q.3.How to print sum of two numbers, let's say 6 and 3

**$ echo 6 + 3**

This will print 6 + 3, not the sum 9, To do sum or math operations in shell use expr, syntax is as follows

Syntax: *expr op1 operator op2*

Where, op1 and op2 are any Integer Number (Number without decimal point) and operator can be

+ Addition
- Subtraction

/ Division
% Modular, to find remainder For e.g. 20 / 3 = 6 , to find remainder 20 % 3 = 2, (Remember its integer calculation)
* Multiplication
**$ expr 6 + 3**
Now It will print sum as 9 , But
**$ expr 6+3**
will not work because space is required between number and operator (See Shell Arithmetic)

Q.4.How to define two variable x=20, y=5 and then to print division of x and y (i.e. x/y)
For Ans. Click here
**$x=20**
**$ y=5**
**$ expr x / y**

Q.5.Modify above and store division of x and y to variable called z
For Ans. Click here
**$ x=20**
**$ y=5**
**$ z=`expr x / y`**
**$ echo $z**

# echo Command

Use echo command to display text or value of variable.

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

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

**Shell Arithmetic**

Use to perform arithmetic operations.

*Syntax:*
expr op1 math-operator op2

*Examples:*
```
$ expr 1 + 3
$ expr 2 - 1
$ expr 10 / 2
$ expr 20 % 3
$ expr 10 \* 3
$ echo `expr 6 + 3`
```

**Note:**
expr 20 %3 - Remainder read as 20 mod 3 and remainder is 2.
expr 10 * 3 - Multiplication use * and not * since its wild card.

For the last statement not the following points

(1) First, before expr keyword we used ` (back quote) sign not the (single quote i.e. ') sign. Back quote is generally found on the key under tilde (~) on PC keyboard OR to the above of TAB key.

(2) Second, expr is also end with ` i.e. back quote.

(3) Here expr 6 + 3 is evaluated to 9, then echo command prints 9 as sum

(4) Here if you use double quote or single quote, it will NOT work
For e.g.
**$ echo "expr 6 + 3"** # It will print expr 6 + 3
**$ echo 'expr 6 + 3'** # It will print expr 6 + 3

**More about Quotes**

There are three types of quotes

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

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

**Exit Status**

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

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

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

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

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

Exercise
Try the following commands and not down the exit status:

```
$ expr 1 + 3
$ echo $?
```

$ echo Welcome
$ echo $?

$ wildwest canwork?
$ echo $?

$ date
$ echo $?

$ echon $?
$ echo $?

**The read Statement**

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

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

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

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

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

| Wild card /Shorthand | Meaning | Examples | |
|---|---|---|---|
| * | Matches any string or group of characters. | **$ ls *** | will show all files |
| | | **$ ls a*** | will show all files whose first name is starting with letter 'a' |
| | | **$ ls *.c** | will show all files having extension .c |
| | | **$ ls ut*.c** | will show all files having extension .c but file name must begin with 'ut'. |
| ? | Matches any single character. | **$ ls ?** | will show all files whose names are 1 character long |
| | | **$ ls fo?** | will show all files |

| | | | |
|---|---|---|---|
| | | | whose names are 3 character long and file name begin with fo |
| **[...]** | Matches any one of the enclosed characters | **$ ls [abc]*** | will show all files beginning with letters a,b,c |

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

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

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

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

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

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

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

**More command on one command line**

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

*Examples:*
**$ date;who**
Will print today's date followed by users who are currently login. Note that You can't use

**$ date who**
for same purpose, you must put semicolon in between date and who command.

**Command Line Processing**

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

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

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

**Exercise**
Try to determine command and arguments from following commands

```
$ ls foo
$ cp y y.bak
$ mv y.bak y.okay
$ tail -10 myf
$ mail raj
$ sort -r -n myf
$ date
$ clear
```

Answer:

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

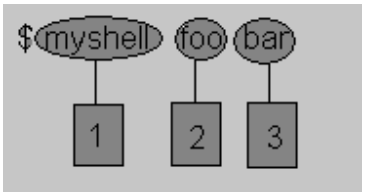
**NOTE:**
**$#** holds number of arguments specified on command line. And **$*** or **$@** refer to all arguments passed to script.

**Why Command Line arguments required**

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

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

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




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 | |

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

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

Mostly all command gives output on screen or take input from keyboard, but in Linux (and in other OS 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**

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

**$cat > sname**
vivek
ashish
zebra
babu
you can *save this file.*

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.

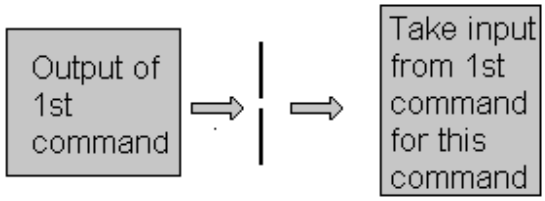
Try one more example to clear your idea:
**$ tr "[a-z]" "[A-Z]" < sname > cap_names**
**$ cat cap_names**
VIVEK
ASHISH
ZEBRA
BABU

**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**

**Pipes**

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




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

*Syntax:*
command1 | command2

*Examles:*

| Command using Pipes | Meaning or Use of Pipes |
|---|---|
| **$ ls \| more** | Output of ls command is given as input to more command So that output is printed one screen full page at a time. |
| **$ who \| sort** | Output of who command is given as input to sort command So that it will print sorted list of users |
| **$ who \| sort > user_list** | Same as above except output of sort is send to (redirected) user_list file |
| **$ who \| wc -l** | Output of who command is given as input to wc command So that it will number of user who logon to system |
| **$ ls -l \| wc -l** | Output of ls command is given as input to wc command So that it will print number of files in current directory. |
| **$ who \| grep raju** | Output of who command is given as input to grep command So that it will print if particular user name if he is logon or nothing is printed (To see particular user is logon or not) |

**Filter**

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

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

Consider one more following example
**$ sort < sname | uniq > u_sname**

Here uniq is filter which takes its input from sort command and passes this lines as input to uniq; Then uniqs output is redirected t

**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.*"

**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<br><br>**$ ps ax \| grep httpd** |
| To see currently running processes and other information like memory and CPU usage with real time updates. | top<br>See the output of top command. | **$ top**<br><br>**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 processes which are created by yourself. An 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 Linux shell programming 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

Answers

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

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

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

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

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

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

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