Institute for Advanced Computing And Software Development (IACSD) Akurdi, Pune

Linux Programming

Dr. D.Y. Patil Educational Complex, Sector 29, Behind Akurdi Railway Station,

Nigdi Pradhikaran, Akurdi, Pune - 411044.

operating system

An operating system can be described as an interface among the computer hardware and the user of any computer. It is a group of software that handles the resources of the computer hardware and facilitates basic services for computer programs.

An operating system is an essential component of system software within a computer system. The primary aim of an operating system is to provide a platform where a user can run any program conveniently or efficiently.

On the other hand, Linux OS is one of the famous versions of the UNIX OS. It is developed to provide a low-cost or free OS for several personal computer system users. Remarkably, it is a complete OS Including an **X Window System, Emacs editor,** IP/TCP, **GUI** (graphical user interface), etc.

Linux

Linux is a computer operating system. An operating system consists of the software that manages your computer and lets you run applications on it. The features that make up Linux and similar computer operating systems include the following:

■ Detecting and preparing hardware — When the Linux system boots up (when you turn on your computer), it looks at the components on your computer (CPU, hard drive, network cards, and so on) and loads the software (drivers and modules) needed to access those particular hardware devices.

■ Managing processes — The operating system must keep track of multiple processes running at the same time and decide which have access to the CPU and when. The system also must offer ways of starting, stopping, and changing the status of processes.

■ Managing memory — RAM and swap space (extended memory) need to be allocated to applications as they need memory. The operating system decides how requests for memory are handled.

■ Providing user interfaces — An operating system must provide ways of accessing the system. The first Linux systems were accessed from a command-line interpreter called the shell. Today, graphical desktop interfaces are commonly available as well.

■ Controlling filesystems — Filesystem structures are built into the operating system (or loaded as modules). The operating system controls ownership and access to the files and directories that the file systems contain.

■ Providing user access and authentication — Creating user accounts and allowing boundaries to be set between users is a basic feature of Linux. Separate user and group accounts enable users to control their own files and processes.

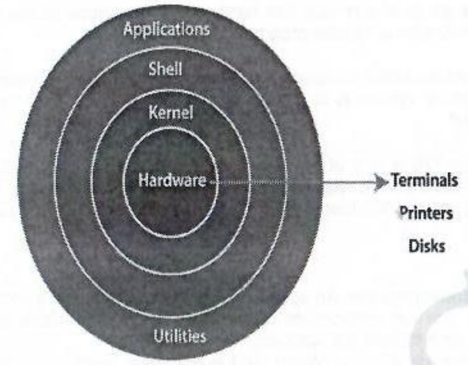
■ Offering administrative utilities — In Linux, there are hundreds (perhaps thousands) of commands and graphical windows to do such things as add users, manage disks, monitor the network, install software, and generally secure and manage your computer.

■ Starting up services — To use printers, handle log messages, and provide a variety of system and network services, processes run in the background, waiting for requests to come in. There are many types of services that run in Linux.

Linux provides different ways of starting and stopping these services. In other words, while Linux includes web browsers to view web pages, it can also be the computer that serves up web pages to others. Popular server features include web, mail, database, printer, fi le, DNS, and DHCP servers.

■ Programming tools — A wide variety of programming utilities for creating applications and libraries for implementing specialty interfaces are available with Linux.

Architecture of Linux system



The Linux operating system's architecture mainly contains some of the components: **the Kernel, System Library, Hardware layer, System,** and **Shell utility**.

1. Kernel:- The kernel is one of the core section of an operating system. It is responsible for each of the major actions of the Linux OS. This operating system contains distinct types of modules and cooperates with underlying hardware directly. The kernel facilitates required abstraction for hiding details of low-level hardware or application programs to the system. There are some of the important kernel types which are mentioned below:

- Monolithic Kernel
- Micro kernels
- Exo kernels
- Hybrid kernels

2. System Libraries:- These libraries can be specified as some special functions. These are applied for implementing the operating system's functionality and don't need code access rights of the modules of kernel.

3. System Utility Programs:- It is responsible for doing specialized level and individual activities.

4. Hardware layer:- Linux operating system contains a hardware layer that consists of several peripheral devices like CPU, HDD, and RAM.

5. Shell:- It is an interface among the kernel and user. It can afford the services of kernel. It can take commands through the user and runs the functions of the kernel. The shell is available in distinct types of OSes. These operating systems are categorized into two different types, which are the **graphical shells** and **command-line shells.**

The graphical line shells facilitate the graphical user interface, while the command line shells facilitate the command line interface. Thus, both of these shells implement operations. However, the graphical user interface shells work slower as compared to the command-line interface shells.

Linux Operating System Features

- **Portable:** Linux OS can perform different types of hardware and the kernel of Linux supports the installation of any type of hardware environment.
- **Open source:** Linux operating system source code is available freely and for enhancing the capability of the Linux OS, several teams are performing in collaboration.
- **Multiprogramming:** Linux OS can be defined as a multiprogramming system. It means more than one application can be executed at the same time.
- **Multi-user:** Linux OS can also be defined as a multi-user system. It means more than one user can use the resources of the system such as **application programs, memory,** or **RAM** at the same time.
- **Hierarchical file system:** Linux OS affords a typical file structure where user files or system files are arranged.
- **Security:** Linux OS facilitates user security systems with the help of various features of authentication such as controlled access to specific files, password protection, or data encryption.
- **Shell:** Linux operating system facilitates a unique interpreter program. This type of program can be applied for executing commands of the operating system. It can be applied to perform various types of tasks such as call application programs and others.

LINUX File system

In Linux file structure files are grouped according to purpose. Ex: commands, data files,

documentation. Parts of a Unix directory tree are listed below. All directories are grouped under the root entry "/".

1. **/ – Root**

Every single file and directory starts from the root directory.

Only root user has write privilege under this directory.

2. **/bin** – User Binaries

Contains binary executables.

Common linux commands you need to use in single-user modes are located under this directory.

Commands used by all the users of the system are located here.

For example: ps, ls, ping, grep, cp.

3. **/sbin** – System Binaries

Just like /bin, /sbin also contains binary executables.

But, the linux commands located under this directory are used typically by system

aministrator, for system maintenance purpose.

For example: iptables, reboot, fdisk, ifconfig, swapon

4. **/etc** – Configuration Files

Contains configuration files required by all programs.

This also contains startup and shutdown shell scripts used to start/stop individual programs.

For example: /etc/resolv.conf, /etc/logrotate.conf

5. **/dev** – Device Files

Contains device files.

These include terminal devices, usb, or any device attached to the system.

For example: /dev/tty1, /dev/usbmon0

6. **/proc** – Process Information

Contains information about system process.

This is a pseudo filesystem contains information about running process. For example: /proc/{pid} directory contains information about the process with that particular pid.

This is a virtual filesystem with text information about system resources. For example: /proc/uptime

7. **/var** – Variable Files

var stands for variable files.

Content of the files that are expected to grow can be found under this directory.

This includes — system log files (/var/log); packages and database files (/var/lib); emails (/var/mail); print queues (/var/spool); lock files (/var/lock); temp files needed across reboots (/var/tmp);

8. **/tmp** – Temporary Files

Directory that contains temporary files created by system and users.

Files under this directory are deleted when system is rebooted.

9. **/usr** – User Programs

Contains binaries, libraries, documentation, and source-code for second level programs.

/usr/bin contains binary files for user programs. If you can't find a user binary under /bin, look under /usr/bin. For example: at, awk, cc, less, scp

/usr/sbin contains binary files for system administrators. If you can't find a system binary under /sbin, look under /usr/sbin. For example: atd, cron, sshd, useradd, userdel

/usr/lib contains libraries for /usr/bin and /usr/sbin

/usr/local contains users programs that you install from source. For example, when you install apache from source, it goes under /usr/local/apache2

10. **/home** – Home Directories

Home directories for all users to store their personal files.

For example: /home/john, /home/dbda

11. **/boot** – Boot Loader Files

Contains boot loader related files.

Kernel initrd, vmlinux, grub files are located under /boot

12. **/lib** – System Libraries

Contains library files that supports the binaries located under /bin and /sbin

Library filenames are either ld* or lib*.so.*

For example: ld-2.11.1.so, libncurses.so.5.7

13. **/opt** – Optional add-on Applications

opt stands for optional.

Contains add-on applications from individual vendors.

add-on applications should be installed under either /opt/ or /opt/ sub-directory.

14. **/mnt** – Mount Directory

Temporary mount directory where sysadmins can mount filesystems.

15. **/media** – Removable Media Devices

Temporary mount directory for removable devices.

For examples, /media/cdrom for CD-ROM; /media/floppy for floppy drives;

/media/cdrecorder for CD writer

16. **/srv** – Service Data

srv stands for service. Contains server specific services related data.

For example, /srv/cvs contains CVS related data.

Access control list and chmod command chown and chgrp commands

1) chmod: Change file access permissions

- **Description:** This command is used to change the file permissions. These permissions read, write and execute permission for the owner, group, and others.
- **Syntax (symbolic mode)**: chmod [ugoa][[+-=][mode]] file
- The first optional parameter indicates who – this can be (u)ser, (g)roup, (o)there, or (a)ll.
- The second optional parameter indicates opcode – this can be for adding (+), removing (-), or assigning (=) permission.
- The third optional parameter indicates the mode – this can be (r)ead, (w)rite, or e(x)ecute.

- Example: Add writes permission for user, group, and others for file1.
- $ chmod ugo+w file1
- **Syntax (numeric mode)**: chmod [mode] file
- The mode is a combination of three digits – the first digit indicates the permission of the user, the second digit for the group, and the third digit for others.
- Each digit is computed by adding the associated permissions. Read permission is '4', write permission is '2' and execute permission is '1'.

- Example: Give read/write/execute permission to the user, read/execute permission to the group, and execute permission to others.
- $ chmod 751 file1

2) chown: Change ownership of the file.
- **Description**: Only the owner of the file has the right to change the file ownership.

- **Syntax**: chown [owner] [file]
- **Example**: Change the owner of file1 to user2 assuming it is currently owned by the current user
  - $ chown user2 file1

3) chgrp: Change the group ownership of the file
- **Description**: Only the owner of the file has the right to change the file ownership

- **Syntax**: chgrp [group] [file]
- **Example**: Change group of file1 to group2 assuming it is currently owned by the current user
  $ chgrp group2 file1

File Handling utilities :

Cat Command: cat linux command concatenates files and print it on the standard output.

SYNTAX:

cat [OPTIONS] [FILE]...

OPTIONS:

-A Show all.

-b Omits line numbers for blank space in the output.

-e A $ character will be printed at the end of each line prior to a new line.

-E Displays a $ (dollar sign) at the end of each line.

-n Line numbers for all the output lines.

-s If the output has multiple empty lines it replaces it with one empty line.

-T Displays the tab characters in the output.

-v Non-printing characters (with the exception of tabs, new-lines and form-feeds) are printed visibly.

Example:

To Create a new file:

cat > file1.txt

This command creates a new file file1.txt. After typing into the file press control+d

(^d) simultaneously to end the file.

1. To Append data into the file: cat >> file1.txt

To append data into the same file use append operator >> to write into the file, else

the file will be overwritten (i.e., all of its contents will be erased).

2. To display a file: cat file1.txt

This command displays the data in the file.

rm COMMAND:

rm linux command is used to remove/delete the file from the directory.

The Syntax is

rm [options..] [file | directory]

OPTIONS:

-f Remove all files in a directory without prompting the user.

-i Interactive. With this option, rm prompts for confirmation before removing any files.

-r (or) -R

Recursively remove directories and subdirectories in the argument list. The directory will be emptied of files and removed. The user is normally prompted for removal of any write-protected files which the directory contains.

EXAMPLE:

1. To Remove / Delete a file:

rm file1.txt

Here rm command will remove/delete the file file1.txt.

2. To delete a directory tree:

rm -ir tmp

This rm command recursively removes the contents of all subdirectories of the tmp directory, prompting you regarding the removal of each file, and then removes the tmp directory itself.

3. To remove more files at once

rm file1.txt file2.txt

rm command removes file1.txt and file2.txt files at the same time.

cd COMMAND:

cd command is used to change the directory.

The Syntax is

cd [directory | ~ | ./ | ../ | -]

OPTIONS:

-L Use the physical directory structure.

-P Forces symbolic links.

EXAMPLE:

1. cd linux-command

This command will take you to the sub-directory(linux-command) from its parent directory.

2. cd ..

This will change to the parent-directory from the current working directory/sub-directory.

3. cd ~

This command will move to the user's home directory which is "/home/username".

cp COMMAND:

cp command copy files from one location to another. If the destination is an existing file, then the file is overwritten; if the destination is an existing directory, the file is copied into the directory (the directory is not overwritten).

SYNTAX:

The Syntax is

cp [OPTIONS]... SOURCE DEST

cp [OPTIONS]... SOURCE... DIRECTORY

cp [OPTIONS]... --target-directory=DIRECTORY SOURCE...

OPTIONS:

-a same as -dpR.

--backup[=CONTROL] make a backup of each existing destination file

-b like --backup but does not accept an argument.

-f if an existing destination file cannot be opened, remove it and try again.

-p same as --preserve=mode,ownership,timestamps.

EXAMPLE:

1. ln -s file1.txt file2.txt

Creates a symbolic link to 'file1.txt' with the name of 'file2.txt'. Here inode for 'file1.txt' and 'file2.txt' will be different.

2. ln -s first second

Creates a symbolic link to 'first' with the name of 'second'.

chown COMMAND:

chown command is used to change the owner / user of the file or directory. This is an

admin command, root user only can change the owner of a file or directory.

SYNTAX:

chown [options] newowner filename/directoryname

OPTIONS:

-R Change the permission on files that are in the subdirectories of the directory that you are currently in.

-c Change the permission for each file.

-f Prevents chown from displaying error messages when it is unable to change the ownership of a file.

EXAMPLE:

1. chown hiox test.txt

The owner of the 'test.txt' file is root, Change to new user hiox.

2. chown -R hiox test

The owner of the 'test' directory is root, With -R option the files and subdirectories user also gets changed.

3. chown -c hiox calc.txt

Here change the owner for the specific 'calc.txt' file only.

Security By File Permissions

Chmod Command:

chmod command allows you to alter / Change access rights to files and directories.

File Permission is given for users, group and others as,

1. User

2. Group

3. Others

Permission 000

SYNTAX:

The Syntax is

chmod [options] [MODE] FileName

OPTIONS:

-c Displays names of only those files whose permissions are being changed

-f Suppress most error messages

-R Change files and directories recursively

-v Output version information and exit.

EXAMPLE:

1. To view your files with what permission they are:

ls -alt

This command is used to view your files with what permission they are.

2. To make a file readable and writable by the group and others.

chmod 066 file1.txt

3. To allow everyone to read, write, and execute the file

chmod 777 file1.txt

mkdir COMMAND:

mkdir command is used to create one or more directories.

SYNTAX:

mkdir [options] directories

OPTIONS:

-m Set the access mode for the new directories.

-p Create intervening parent directories if they don't exist.

-v Print help message for each directory created.

EXAMPLE:

1. Create directory:

mkdir test

The above command is used to create the directory 'test'.

2. Create directory and set permissions:

mkdir -m 666 test

The above command is used to create the directory 'test' and set the read and write permission.

rmdir COMMAND:

rmdir command is used to delete/remove a directory and its subdirectories.

SYNTAX:

The Syntax is

rmdir [options..] Directory

OPTIONS:

-p Allow users to remove the directory dirname and its parent directories which become empty.

EXAMPLE:

1. To delete/remove a directory

rmdir tmp

rmdir command will remove/delete the directory tmp if the directory is empty.

2. To delete a directory tree:

rm -ir tmp

This command recursively removes the contents of all subdirectories of the tmp directory, prompting you regarding the removal of each file, and then removes the tmp directory itself.

mv COMMAND:

mv command which is short for move. It is used to move/rename file from one directory to another. mv command is different from cp command as it completely removes the file from the source and moves to the directory specified, where cp command just copies the content from one file to another.

SYNTAX:

The Syntax is

mv [-f] [-i] oldname newname

find command- To find a file in Linux, you can use the Linux find command. This starts a **recursive search**, where a directory hierarchy is searched following certain criteria. The Linux find command is a precise tool for finding files and directories and is supported across pretty much all Linux distributions.

general structure of the Linux find command:

find <directory_path> <search_parameter>

Search parameter	Explanation
-name, -iname	Filter by file name
-type	Filter by file type
-size, -empty	Filter by file size
-ctime, -mtime, -atime	Filter by time stamp
-user, -group	Filter by owner and group
-perm	Filter by file rights

Diff command - diff stands for difference. This command is used to display the differences in the files by comparing the files line by line.
diff [options] File1 File2

fsck
The fsck (File System Consistency Check) Linux utility checks filesystems for errors or outstanding issues. The tool is used to fix potential errors and generate reports.
This utility comes by default with Linux distributions. No specific steps or an installation procedure is required to use fsck. Once you load the terminal, you are ready to exploit the functionalities of the tool.
fsck <options> <filesystem>
example : sudo fsck /dev/sdb
cut command
It is used to cut a specific sections by byte position, character, and field and writes them to standard output. It cuts a line and extracts the text data. It is necessary to pass an argument with it; otherwise, it will throw an error message.
cut a specific section, it is necessary to specify the delimiter. A delimiter will decide how the sections are separated in a text file. Delimiters can be a space (' '), a hyphen (-), a slash (/), or anything else. After '-f' option, the column number is mentioned.
cut OPTION... [FILE]...
To cut by using the hyphen (-) as the delimiter, execute the below command:
cut -d- -f2 marks.txt

sed command

Linux 'sed' command stands for stream editor. It is used to edit streams (files) using regular expressions. But this editing is not permanent. It remains only in display, but in actual, file content remains the same.

SED is a powerful text stream editor. Can do insertion, deletion, search and replace(substitution). SED command in unix supports regular expression which allows it perform complex pattern matching.

Syntax: sed OPTIONS... [SCRIPT] [INPUTFILE...]

Replacing or substituting string : Sed command is mostly used to replace the text in a file. The below simple sed command replaces the word "unix" with "linux" in the file.

```
$sed 's/unix/linux/' geekfile.txt
```

Sort command

The 'sort' command sorts the file content in an alphabetical order.

Create a file mix.txt

Command :

```
$ cat > mix.txt
time
save
apple
manual
$ sort mix.txt
apple
manual
save
time
```

ssh command

This command is used to start the SSH client program that enables secure connection to the SSH server on a remote machine. The ssh command is used from logging into the remote machine, transferring files between the two machines, and for executing commands on the remote machine.

```
ssh user_name@host(IP/Domain_name)
ssh dbda@10.143.90.2
```

The ssh command is often also used to remotely execute commands on the remote machine without logging in to a shell prompt. The syntax for this is:

```
ssh hostname command
```

For example, to execute the command:

```
ls /tmp/doc
```

on host sample.ssh.com, type the following command at a shell prompt:

```
ssh sample.ssh.com ls /tmp/doc
```

After authenticating to the remote server, the contents of the remote directory will be displayed, and you will return to your local shell prompt. -x Disables X11 forwarding.

Systemctl command

The systemctl command is a powerful tool in Linux that enables system administrators to manage system services and units. It is used to start, stop, restart, enable, disable, and reload services and daemons in the system.

In this article, we will explore the syntax, options, and usage of the systemctl command.

```
systemctl [command] [service]
```

Starting Services

If we want to start our SSH service.

```
systemctl start sshd
```

Gzip Command

gzip command compresses files. If given a file as an argument, gzip compresses the file, adds a ".gz" suffix, and deletes the original file. With no arguments, gzip compresses the standard input and writes the compressed file to standard output.

```
$ gzip mydoc.txt
```

gunzip command is used to compress or expand a file or a list of files in Linux. It accepts all the files having extension as **.gz**, **.z**, **_z**, **-gz**, **-z** , **.Z**, **.taz** or**.tgz** and replace the compressed file with the original file by default. The files after uncompression retain its actual extension.

Syntax:

```
gunzip [Option] [archive name/file name]
```

Zip command is used to compress files to reduce file size and is also used as a file package utility. Zip is available in many operating systems like Unix, Linux, windows, etc.

If you have limited bandwidth between two servers and want to transfer the files faster, then zip the files and transfer them.

```
zip [file_name.zip] [file_name]
```

Using the simple unzip command, you can extract all files from the zip archive into the current zip file directory as follows:

```
$ unzip filename.zip
```

For example, we have downloaded a zip file in the 'Downloads' with the name 'testfile.zip'. So, first, navigate into Downloads directory and then we have extracted the zip file using the following command:

```
$ cd Downloads
$ unzip testfile.zip
```

Environment Variables command

echo $VARIABLE - To display value of a variable

env - Displays all environment variables

variable_name=var_value - Create a new variable

tar command

The Linux 'tar' stands for tape archive, which is used to create Archive and extract the Archive files. tar command in Linux is one of the important commands which provides archiving functionality in Linux. We can use the Linux tar command to create compressed or uncompressed Archive files and also maintain and modify them.

Syntax of `tar` command in Linux

```
tar [options] [archive-file] [file or directory to be archived]
```

touch command is a common Linux command to make an empty file or modify the file timestamps.

```
touch <filename>
```

who command

The Linux "who" command lets you display the users currently logged in to your UNIX or Linux operating system.

whoami Command

allows you to know the username associated with the current effective user ID

kill command

In Linux, the kill command (located inside the /bin/kill) can be defined as a built-in command.

It is used for manually terminating the processes.

The tput command

The tput command outputs a string if the attribute Capability Name is of type string. The output string is an integer if the attribute is of type integer. If the attribute is of type Boolean, the tput command sets the exit value (0 for TRUE, 1 for FALSE), and produces no other output.
Syntax - tput [-TType] [CapabilityName [Parameters...]]
For Using stdin to Process Multiple Capabilities

tput [-S]

Description

The **tput** command uses the terminfo database to make terminal-dependent information available to the shell. The **tput** command outputs a string if the attribute CapabilityName is of type string. The output string is an integer if the attribute is of type integer. If the attribute is of type Boolean, the **tput** command sets the exit value (0 for TRUE, 1 for FALSE), and produces no other output.

Grep command

Grep is a useful command to search for matching patterns in a file. grep is short for "global regular expression print".

grep [OPTION...] PATTERNS [FILE...]

In the above syntax, grep searches for PATTERNS in each FILE. Grep finds each line that matched the provided PATTERN. It is a good practice to close the PATTERN in quotes when grep is used in a shell command.

following options that can be used with grep:

- -i, --ignore-case: Ignores case distinctions in patterns and input data.
- -v, --invert-match: Selects the non-matching lines of the provided input pattern.
- -n, --line-number: Prefix each line of the matching output with the line number in the input file.
- -w: Find the exact matching word from the input file or string.
- -c: Count the number of occurrences of the provided pattern.

Example

grep -i "berries" fruits.txt

We can command grep to return results while ignoring the case of the matching string. Let's find the word "berries" from our sample file. It should match all occurrences of "berries" regardless of their case.

Networking commands:
These are most useful commands while working on Linux server , this enables you to quickly troubleshoot connection issues e.g. whether other system is connected or not , whether other host is responding or not .

let's see some example of various networking command in Unix and Linux. Some of them are quite basic e.g. ping and telnet and some are more powerful e.g. nslookup and netstat. When you used these commands in combination of find and grep you can get anything you are looking for e.g. hostname, connection end points, connection status etc.
hostname
hostname with no options displays the machines host name
hostname –d displays the domain name the machine belongs to
hostname –f displays the fully qualified host and domain name
hostname –i displays the IP address for the current machine
ping
It sends packets of information to the user-defined source. If the packets are received, the destination device sends packets back. Ping can be used for two purposes

Example : ping 127.0.0.1
1. To ensure that a network connection can be established.
2. Timing information as to the speed of the connection.
If you do ping www.yahoo.com it will display its IP address. Use ctrl+C to stop the test.
ifconfig
View network configuration, it displays the current network adapter configuration. It is handy to determine if you are getting transmit (TX) or receive (RX) errors.
netstat
Most useful and very versatile for finding connection to and from the host. You can find out all the multicast groups (network) subscribed by this host by issuing "netstat -g"
netstat -nap | grep port will display process id of application which is using that port
netstat -a or netstat –all will display all connections including TCP and UDP
netstat --tcp or netstat –t will display only TCP connection
netstat --udp or netstat –u will display only UDP connection
netstat -g will display all multicast network subscribed by this host.
nslookup
If you know the IP address it will display hostname. To find all the IP addresses for a given domain name, the command nslookup is used. You must have a connection to the internet for this utility to be useful.
E.g. nslookup blogger.com
You can also use nslookup to convert hostname to IP Address and from IP Address from hostname.
traceroute
A handy utility to view the number of hops and response time to get to a remote system or website is traceroute. Again you need an internet connection to make use of this tool.
finger View user information, displays a user's login name, real name, terminal name and write

status.
telnet
Connects destination host via telnet protocol, if telnet connection establish on any port means connectivity between two hosts is working fine.
telnet hostname port will telnet hostname with the port specified.

To use telnet command to log in to a server, use the syntax below.

```
$ telnet 30.34.11.9
```

VI Text Editor

First, you need to launch the VI editor to begin working on it. To launch the editor, open your Linux terminal and then type:

```
vi <filename_NEW> or <filename_EXISTING>
```

And if you mention an existing file, VI would open it to edit. Alternatively, you're free to create a completely new file.

VI Editing Commands

You need to be in the command mode to run editing commands in the VI editor. VI is case-sensitive. Hence, make sure you use the commands in the correct letter case. Also, make sure you type the right command to avoid undesired changes. Below are some of the essential commands to use in VI.

i – Inserts at cursor (gets into the insert mode)

a – Writes after the cursor (gets into the insert mode)

A – Writes at the ending of a line (gets into the insert mode)

o – Opens a new line (gets into the insert mode)

ESC – Terminates the insert mode

u – Undo the last change

U – Undo all changes of the entire line

D – Deletes the content of a line after the cursor

R – Overwrites characters from the cursor onwards

r – Replaces a character

s – Substitutes one character under the cursor and continue to insert

S – Substitutes a full line and start inserting at the beginning of a line

~ – Changes a character's case

dd – Deletes the line

3dd – Deletes 3 lines

dw – Deletes a word

4dw – Deletes 4 words

x – Deletes a character at the cursor

cw – Changes the word

Commands for Moving within a File

For moving within a file, you should be in the command mode. Also, arrow keys can be used to navigate. Below are the commands used to navigate within a file.

k – Moves cursor up

j – Moves cursor down

l – Moves cursor right

h – Moves cursor left

Commands for Saving and Closing a File

To exit the text editor and save the changes to a file, you need to be in the command mode. Below are the commands to use for saving and closing a file in the editor.

Shift+zz – Saves a file and exits

:w – Saves a file and keeps the file open

:q – Exits without saving

:wq – Saves a file and quits

Shell Programming

A bash script is a file containing a sequence of commands that are executed by the bash program line by line. It allows you to perform a series of actions, such as navigating to a specific directory, creating a folder, and launching a process using the command line.

By saving these commands in a script, you can repeat the same sequence of steps multiple times and execute them by running the script.

Overview of Bash shell and command line interface

The shell accepts commands from the user and displays the output
Although Bash is a type of shell, there are other shells available as well, such as Korn shell (ksh), C shell (csh), and Z shell (zsh). Each shell has its own syntax and set of features, but they all share the common purpose of providing a command-line interface for interacting with the operating system.

You can determine your shell type using the ps command:
ps

Running Bash commands from the command line

As mentioned earlier, the shell prompt looks something like this:

[username@host ~]$
You can enter any command after the $ sign and see the output on the terminal.
Generally, commands follow this syntax:

command [OPTIONS] arguments
Let's discuss a few basic bash commands and see their outputs.

date: Displays the current date

- pwd: Displays the present working directory.
  /home/user/shell-tutorial
- ls: Lists the contents of the current directory.
  user@User:~/shell-tutorial$ ls
  check_plaindrome.sh count_odd.sh env log temp
- echo: Prints a string of text, or value of a variable to the terminal.
  user@User:~/shell-tutorial$ echo "Hello bash"
  Hello bash
  You can always refer to a commands manual with the man command.
  You can see options for a command in detail using man

How to Create and Execute Bash scripts

Script naming conventions

By naming convention, bash scripts end with .sh. However, bash scripts can run perfectly fine without the sh extension.

Adding the Shebang

Bash scripts start with a shebang. Shebang is a combination of bash # and bang ! followed by the bash shell path. This is the first line of the script. Shebang tells the shell to execute it via bash shell. Shebang is simply an absolute path to the bash interpreter.
Below is an example of the shebang statement.

#!/bin/bash
You can find your bash shell path (which may vary from the above) using the command:

which bash

Arithmetic in shell script –

The following arithmetic operators are supported by Bourne Shell.

all the conditional expressions should be inside square braces with spaces around them, for example **[$a == $b]** is correct whereas, **[$a==$b]** is incorrect.

Assume variable **a** holds 10 and variable **b** holds 20 then –

Operator	**Description**	**Example**
+ (Addition)	Adds values on either side of the operator	`expr $a + $b` will give 30
- (Subtraction)	Subtracts right hand operand from left hand operand	`expr $a - $b` will give -10
* (Multiplication)	Multiplies values on either side of the operator	`expr $a \* $b` will give 200
/ (Division)	Divides left hand operand by right hand operand	`expr $b / $a` will give 2
% (Modulus)	Divides left hand operand by right hand operand and returns remainder	`expr $b % $a` will give 0
= (Assignment)	Assigns right operand in left operand	a = $b would assign value of b into a
== (Equality)	Compares two numbers, if both are same then returns true.	[$a == $b] would return false.
!= (Not Equality)	Compares two numbers, if both are different then returns true.	[$a != $b] would return true.

Creating our first bash script

Our first script prompts the user to enter a path. In return, its contents will be listed.

Create a file named run_all.sh using the nano command. You can use any editor of your choice.
nano run_all.sh
Add the following commands in your file and save it:

```
#!/bin/bash
echo "Today is " `date`

echo -e "\nenter the path to directory"
read the_path

echo -e "\n you path has the following files and folders: "
ls $the_path
```
Script to print contents of a user supplied directory

Executing the bash script

To make the script executable, assign execution rights to your user using this command:

```
chmod u+x run_all.sh
```

Here,

- chmod modifies the ownership of a file for the current user :u.
- +x adds the execution rights to the current user. This means that the user who is the owner can now run the script.
- run_all.sh is the file we wish to run.
  You can run the script using any of the mentioned methods:

- sh run_all.sh
- bash run_all.sh
- ./run_all.sh

Comments in bash scripting

Comments start with a # in bash scripting. This means that any line that begins with a # is a comment and will be ignored by the interpreter.
Comments are very helpful in documenting the code, and it is a good practice to add them to help others understand the code.

These are examples of comments:

```
# This is an example comment
```

Variables and data types in Bash

Variables let you store data. You can use variables to read, access, and manipulate data throughout your script.

There are no data types in Bash. In Bash, a variable is capable of storing numeric values, individual characters, or strings of characters.

In Bash, you can use and set the variable values in the following ways:

1. Assign the value directly:

```
country=India
```

2. Assign the value based on the output obtained from a program or command, using command substitution. Note that $ is required to access an existing variable's value.

```
same_country=$country
```

This assigns the value of countryto the new variable same_country
To access the variable value, append $ to the variable name.

```
user@User:~$ country=India
user@User:~$ echo $country
India
user@User:~$ new_country=$country
user@User:~$ echo $new_country
India
```

Input and output in Bash scripts

Gathering input

In this section, we'll discuss some methods to provide input to our scripts.

1. Reading the user input and storing it in a variable

We can read the user input using the read command.

```
#!/bin/bash
echo "Today is " `date`

echo -e "\nenter the path to directory"
read the_path

echo -e "\nyour path has the following files and folders: "
ls $the_path
```

2. Reading from a file

This code reads each line from a file named input.txt and prints it to the terminal. We'll study while loops later in this article.

```
while read line
do
  echo $line
done < input.txt
```

3. Command line arguments

In a bash script or function, $1 denotes the initial argument passed, $2 denotes the second argument passed, and so forth.
This script takes a name as a command-line argument and prints a personalized greeting.

```
echo "Hello, $1!"
```

We have supplied User as our argument to the script.

```
#!/bin/bash
echo "Hello, $1!"
```

The code for the script: greeting.sh

Displaying output

Here we'll discuss some methods to receive output from the scripts.

1. Printing to the terminal:

```
echo "Hello, World!"
```

This prints the text "Hello, World!" to the terminal.

2. Writing to a file:

```
echo "This is some text." > output.txt
```

This writes the text "This is some text." to a file named output.txt. Note that the >operator overwrites a file if it already has some content.

3. Appending to a file:

```
echo "More text." >> output.txt
```

This appends the text "More text." to the end of the file output.txt.

4. Redirecting output:

```
ls > files.txt
```

This lists the files in the current directory and writes the output to a file named files.txt. You can redirect output of any command to a file this way.

Basic Bash commands (echo, read, etc.)

Here is a list of some of the most commonly used bash commands:

1. cd: Change the directory to a different location.
2. ls: List the contents of the current directory.
3. mkdir: Create a new directory.
4. touch: Create a new file.
5. rm: Remove a file or directory.
6. cp: Copy a file or directory.
7. mv: Move or rename a file or directory.
8. echo: Print text to the terminal.
9. cat: Concatenate and print the contents of a file.
10. grep: Search for a pattern in a file.
11. chmod: Change the permissions of a file or directory.
12. sudo: Run a command with administrative privileges.
13. df: Display the amount of disk space available.
14. history: Show a list of previously executed commands.
15. ps: Display information about running processes.

Conditional statements (if/else)

Expressions that produce a boolean result, either true or false, are called conditions. There are several ways to evaluate conditions, including if, if-else, if-elif-else, and nested conditionals.

Syntax:

```
if [[ condition ]];
then
        statement
elif [[ condition ]]; then
        statement
else
        do this by default
fi
```

Syntax of bash conditional statements

We can use logical operators such as AND -a and OR -o to make comparisons that have more significance.

```
if [ $a -gt 60 -a $b -lt 100 ]
```

This statement checks if both conditions are true: a is greater than 60 AND b is less than 100.

Let's see an example of a Bash script that uses if, if-else, and if-elif-else statements to determine if a user-inputted number is positive, negative, or zero:

```
#!/bin/bash

echo "Please enter a number: "
read num

if [ $num -gt 0 ]; then
```

```
  echo "$num is positive"
elif [ $num -lt 0 ]; then
  echo "$num is negative"
else
  echo "$num is zero"
fi
```

Script to determine if a number is positive, negative, or zero

The script first prompts the user to enter a number. Then, it uses an if statement to check if the number is greater than 0. If it is, the script outputs that the number is positive. If the number is not greater than 0, the script moves on to the next statement, which is an if-elif statement. Here, the script checks if the number is less than 0. If it is, the script outputs that the number is negative. Finally, if the number is neither greater than 0 nor less than 0, the script uses an else statement to output that the number is zero.

Looping and Branching in Bash(for , while , until)

For loop

The for loop, just like the while loop, allows you to execute statements a specific number of times. Each loop differs in its syntax and usage.

In the example below, the loop will iterate 5 times.

```
#!/bin/bash

for i in {1..5}
do
   echo $i
done
```

For loop that iterates 5 times.

While loop

While loops check for a condition and loop until the condition remains true. We need to provide a counter statement that increments the counter to control loop execution.

In the example below, ((i += 1)) is the counter statement that increments the value of i. The loop will run exactly 10 times.

```
#!/bin/bash
i=1
while [[ $i -le 10 ]] ; do
  echo "$i"
  (( i += 1 ))
done
```

While loop that iterates 10 times.

Until Loop

in a bash scripting is used to execute a set of commands repeatedly based on the boolean result of an expression. The set of commands are executed only until the expression evaluates to true. It means that when the expression evaluates to false, a set of commands are executed iteratively. The loop is terminated as soon as the expression evaluates to true for the first time.

Example :

```
1. #!/bin/bash
2. #Bash Until Loop example with a single condition
3.
```

```
4. i=1
5. until [ $i -gt 10 ]
6. do
7. echo $i
8. ((i++))
9. done
```

Case statements

In Bash, case statements are used to compare a given value against a list of patterns and execute a block of code based on the first pattern that matches. The syntax for a case statement in Bash is as follows:

```
case expression in
  pattern1)
    # code to execute if expression matches pattern1
    ;;
  pattern2)
    # code to execute if expression matches pattern2
    ;;
  pattern3)
    # code to execute if expression matches pattern3
    ;;
  *)
    # code to execute if none of the above patterns match expression
    ;;
esac
```

Case statements syntax

Here, "expression" is the value that we want to compare, and "pattern1", "pattern2", "pattern3", and so on are the patterns that we want to compare it against.

The double semicolon ";;" separates each block of code to execute for each pattern. The asterisk "*" represents the default case, which executes if none of the specified patterns match the expression.

Let's see an example.

```
fruit="apple"

case $fruit in
  "apple")
    echo "This is a red fruit."
    ;;
  "banana")
    echo "This is a yellow fruit."
    ;;
  "orange")
    echo "This is an orange fruit."
    ;;
  *)
    echo "Unknown fruit."
    ;;
```

```
esac
```

Example of case statement

In this example, since the value of "fruit" is "apple", the first pattern matches, and the block of code that echoes "This is a red fruit." is executed. If the value of "fruit" were instead "banana", the second pattern would match and the block of code that echoes "This is a yellow fruit." would execute, and so on. If the value of "fruit" does not match any of the specified patterns, the default case is executed, which echoes "Unknown fruit."

break and **continue** commands-

allow you to exit a loop early or skip the remaining commands in the loop and return to the beginning.

Both the **break** and the **continue** commands are meant to be used only in **for, while** and **until** loops. In fact, if you try to invoke the **break** command on its own, bash will tell you just that.

The **break** command exits the loop and moves to whatever command follows it.

The **continue** command works similarly to **break**, but it doesn't exit the loop unless it's on its last pass through it. Otherwise, it skips the remaining commands in the loop and returns to the top to continue.

Shell metacharacters

The most powerful feature of the Linux Bash shell is its capability to work around files and

redirect their input and output efficiently. Linux uses special characters or symbols known as metacharacters that add special meaning to a shell command with respect to file search and commands connection.

The metacharacters are helpful in listing, removing, and copying files on Linux. However, the function of each metacharacter differs depending on the command you are using it with.

These are some file-matching metacharacters that the Linux shell can interpret:

*** (Asterisk):** Matches single or multiple occurrences of a character

? (Question mark): Matches a single character or a pattern occurrence

[ ] (Square Brackets): Matches any hyphen-separated number, symbol, or alphabets

specified inside the squared brackets

The Linux file redirection metacharacters allow you to redirect the content to (>) and from (<) the files.

The three primary redirection metacharacters are:

1. <: Directs the file content to the command. For instance, the command output for less .bashrc is the same as less < .bashrc.

2. >: Directs the command output to the file. The command ls /etc > lists.txt saves the

output to the lists.txt file.

3. >>: Appends the command output to the file content.

The brace expansion metacharacter allows you to expand the characters across directories, file names, or other command-line arguments. For instance, you can make a new directory brace inside the /tmp folder and create a set of files using the touch command as follows:

sudo mkdir /tmp/brace; cd /tmp/brace

Command line expansion -

When a command is entered in the command line, it expands into its output which is displayed. This is called expansion. The command you're typing will be printed with the help of echo command on the terminal.

set -x command Used to enable shell expansion.

set +x command Used to enable shell expansion.

Directory stacks manipulation –

A directory stack is a stack in which we can push and pop directories when we need them.

The directory stack is based on the principle of LIFO. What is LIFO? LIFO means last in, first out. The last directory pushed in the stack will be the first directory which will go out of the stack. To play with the stack directory you need to learn and use the following commands:

dirs

dirs [+N | -N] [-clpv]

Display the list of currently remembered directories. Directories are added to the list with the pushd command; the popd command removes directories from the list.

+N

Displays the Nth directory (counting from the left of the list printed by dirs when invoked without options), starting with zero.

-N

Displays the Nth directory (counting from the right of the list printed by dirs when invoked without options), starting with zero.

-c

Clears the directory stack by deleting all of the elements.

-l

Produces a longer listing; the default listing format uses a tilde to denote the home directory.

-p

Causes dirs to print the directory stack with one entry per line.

-v

Causes dirs to print the directory stack with one entry per line, prefixing each entry with its index in the stack.

popd

popd [+N | -N] [-n]

Remove the top entry from the directory stack, and cd to the new top directory. When no arguments are given, popd removes the top directory from the stack and performs a cd to the new top directory. The elements are numbered from 0 starting at the first directory listed with dirs; i.e., popd is equivalent to popd +0.

+N

Removes the Nth directory (counting from the left of the list printed by dirs), starting with zero.

-N

Removes the Nth directory (counting from the right of the list printed by dirs), starting with zero.

-n

Suppresses the normal change of directory when removing directories from the stack, so that only the stack is manipulated.

pushd

pushd [dir | +N | -N] [-n]

Save the current directory on the top of the directory stack and then cd to dir. With no arguments, pushd exchanges the top two directories.

+N

Brings the Nth directory (counting from the left of the list printed by dirs, starting with zero) to the top of the list by rotating the stack.

-N

Brings the Nth directory (counting from the right of the list printed by dirs, starting with zero) to the top of the list by rotating the stack.

-n

Suppresses the normal change of directory when adding directories to the stack, so that only the stack is manipulated.

dir

Makes the current working directory be the top of the stack, and then executes the equivalent of `cd dir'. cds to dir.

Example –

```
$ pushd songs
   ~/ songs  ~  /home
4  $ dirs
   ~/ songs ~ /home
5  $ dirs -l
   /home/ellie/ songs /home/ellie  /home
6  $ popd
   ~/home
```