Lab No 02.b

Linux Directory Structure

Objective:

Objective of this lab is to demonstrate directory structure of Linux system

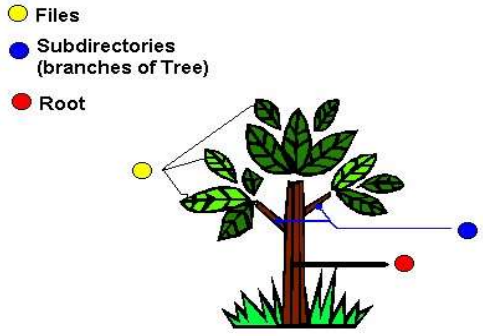
Scope:

Linux system

A file system is a logical collection of files on a partition or disk. A partition is a container for information and can span an entire hard drive if desired.

Directory Structure

Unix uses a hierarchical file system structure, much like an upside-down tree, with root (/) at the base of the file system and all other directories spreading from there.



Linux File System is just like a tree

/ This is the root directory which should contain only the directories needed at the top level of the file structure
Everything on your Linux system is located under the / directory, known as the root directory. You can think of the / directory as being similar to the C:\ directory on Windows – but this isn't strictly true, as Linux doesn't have drive letters.

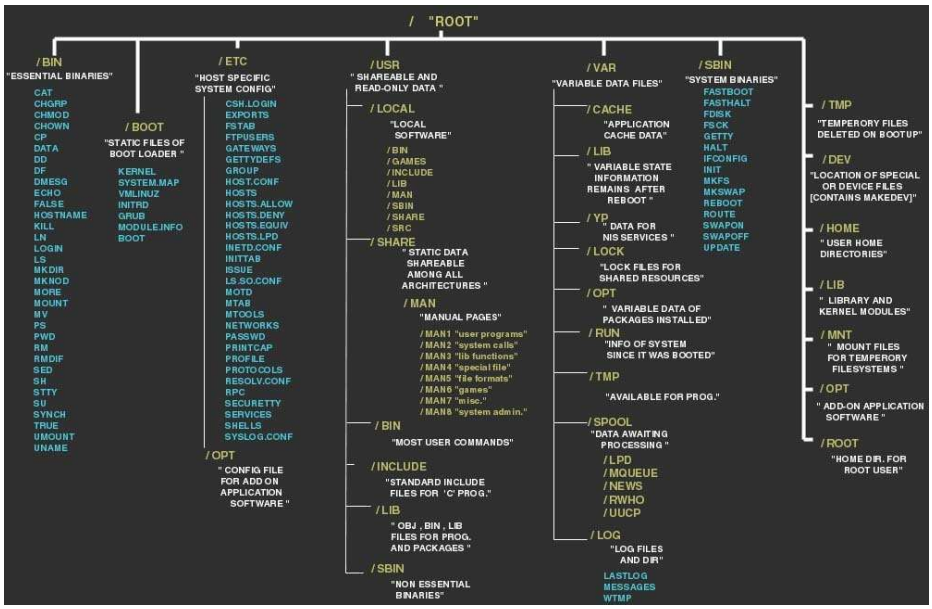
/bin This is where the executable files are located. These files are available to all users. The /bin directory contains the essential user binaries (programs) that must be present when the system is mounted in single-user mode. Applications such as Firefox are stored in /usr/bin, while important system programs and utilities such as the bash shell are located in /bin.

/dev These are device drivers
Linux exposes devices as files, and the /dev directory contains a number of special files that represent devices. These are not actual files as we know them, but they appear as files – for example, /dev/sda represents the first SATA drive in the system. If you wanted to partition it, you could start a partition editor and tell it to edit /dev/sda.

/boot The /boot directory contains the files needed to boot the system – for example, the GRUB boot loader's files and your Linux kernels are stored here. The boot loader's configuration files aren't located here, though – they're in /etc with the other configuration files.

/etc The /etc directory contains configuration files, which can generally be edited by hand in a text editor. Note that the /etc/ directory contains system-wide configuration files – user-specific configuration files are located in each user's home directory.

/home	The /home directory contains a home folder for each user. For example, if your user name is bob, you have a home folder located at /home/bob. This home folder contains the user's data files and user-specific configuration files. Each user only has write access to their own home folder and must obtain elevated permissions (become the root user) to modify other files on the system.
/lib	The /lib directory contains libraries needed by the essential binaries in the /bin and /sbin folder. Libraries needed by the binaries in the /usr/bin folder are located in /usr/lib.
/lost+found	Each Linux file system has a lost+found directory. If the file system crashes, a file system check will be performed at next boot. Any corrupted files found will be placed in the lost+found directory, so you can attempt to recover as much data as possible.
/media	The /media directory contains subdirectories where removable media devices inserted into the computer are mounted. For example, when you insert a CD into your Linux system, a directory will automatically be created inside the /media directory. You can access the contents of the CD inside this directory.
/mnt	Used to mount other temporary file systems, such as cdrom and floppy for the CD-ROM drive and floppy diskette drive, respectively
/opt	he /opt directory contains subdirectories for optional software packages. It's commonly used by proprietary software that doesn't obey the standard file system hierarchy – for example, a proprietary program might dump its files in /opt/application when you install it.
/proc	The /proc directory similar to the /dev directory because it doesn't contain standard files. It contains special files that represent system and process information.
/root	The /root directory is the home directory of the root user. Instead of being located at /home/root, it's located at /root. This is distinct from /, which is the system root directory.
/tmp	Holds temporary files used between system boots
/usr	Used for miscellaneous purposes, and can be used by many users. Includes administrative commands, shared files, library files, and others
/var	Typically contains variable-length files such as log and print files and any other type of file that may contain a variable amount of data
/sbin	Contains binary (executable) files, usually for system administration. For example, fdisk and ifconfig utlities



Homework:

1. Understand linux directory system.
2. Use BASH Commands on terminal
3. Use Help command to explore more commands
4. Use Command with --help parameter to explore more options
5. Use man command to explore manual about any command.

Lab No 03.a

BASH Scripting

Objective: Objective of this Lab is to understand how to write a LInux script
Scope: Linux Bash

Step 1: Choose Text Editor

Shell scripts are written using text editors. On Linux systems, there are a number to choose from: Vim, Emacs, Nano, Pico, Kedit, Gedit, Geany, Notepad++, Kate, Jed or LeafPad.

Once you have chosen a text editor, start the text editor, open a new file to begin to typing a shell script.

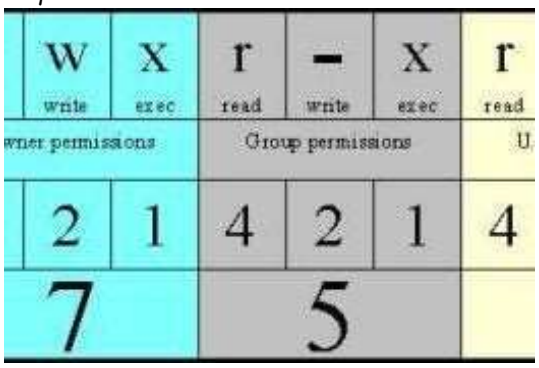
Step 2: Type in Commands

Start to type in basic commands that you would like the script to run.
Be sure to type each command on a separate line.
For example, to print out words to the screen use the "echo" command:
echo "This statement will print out to the screen."
To list files in a directory, type:
echo "Now we are going to list files."
ls
To print the current directory you are in, type:
echo "Next we are going to print the directory we are in:"
pwd
Save the file under the name: FirstShellScript.sh

Step 3: Make File Executable

Now that the file has been saved, it needs to be made executable. This is done using the chmod command. On your Linux command line type:

chmod 555 FirstShellScript.sh



This will allow you to execute the shell script to run the commands contained within it.

Step 4: Run the Shell Script

1. To run the shell script, navigate to the directory where the file you just saved exists.

2. Now type the following [be sure to type the "dot slash" before it!]:

./FirstShellScript.sh

3. Then hit the Enter key to execute it

4. The commands that you saved in the shell script will now run.

Lab No 03.b

Variables

Objective: To Understande the basics of BASH

1. SHELL Keywords

echo, read, if fi, else, case, esac, for , while , do , done, until , set, unset, readonly, shift, export, break, continue, exit, return, trap , wait, eval ,exec, ulimit , umask.

2. General things SHELL

The shbang line

The "shbang" line is the very first line of the script and lets the kernel know what shell will be interpreting the lines in the script. The shbang line consists of a #! followed by the full pathname to the shell, and can be followed by options to control the behavior of the shell.

EXAMPLE

```
#!/bin/sh
```

Comments

Comments are descriptive material preceded by a # sign. They are in effect until the end of a line and can be started anywhere on the line.

EXAMPLE

```
# this text is not
# interpreted by the shell
```

Wildcards

There are some characters that are evaluated by the shell in a special way. They are called shell metacharacters or "wildcards." These characters are neither numbers nor letters. For example, the *, ?, and [ ] are used for filename expansion. The <, >, 2>, >>, and | symbols are used for standard I/O redirection and pipes. To prevent these characters from being interpreted by the shell they must be quoted.

3. SHELL Variables

Shell variables change during the execution of the program.

For example:

myname= “Salman”

myname = "Salman Ahmed"

age=25

A $ sign operator is used to recall the variable values.

For example:

echo $myname will display Salman Ahmed on the screen.

Local variables

Local variables are in scope for the current shell. When a script ends, they are no longer available; i.e., they go out of scope. Local variables are set and assigned values.

EXAMPLE

```
variable_name=value
name="John Doe"
x=5
```

Global variables

Global variables are called environment variables. They are set for the currently running shell and any process spawned from that shell. They go out of scope when the script ends.

EXAMPLE

```
VARIABLE_NAME=value
export VARIABLE_NAME
PATH=/bin:/usr/bin:.
export PATH
```

Extracting values from variables To extract the value from variables, a dollar sign is used.

EXAMPLE

echo $variable_name
echo $name
echo $PATH

Rules

1. A variable name is any combination of alphabets, digits and an underscore („-„);
2. No commas or blanks are allowed within a variable name.
3. The first character of a variable name must either be an alphabet or an underscore.
4. Variables names should be of any reasonable length.
5. Variables name are case sensitive . That is , Name, NAME, name, Name, are all different variables.

4. **Expression Command**

To perform all arithematic operations .
Syntax :
Var = $value1‟ + $ value2‟
To perform all String operations
EXAMPLE

Equality:

=	string
!=	string
-eq	number
-ne	number

Logical:

-a	and
-o	or
!	not

Logical:

AND	&&
OR	||

05.READ Statement :

To get the input from the user.
Syntax :
read x y
(no need of commas between variables)

06. ECHO Statement :

Similar to the output statement. To print output to the screen, the echo command is used.
Wildcards must be escaped with either a backslash or matching quotes.
Syntax :
Echo “String” (or) echo $ b(for variable).
EXAMPLE
echo "What is your name?"
Reading user input
The read command takes a line of input from the user and assigns it to
a variable(s) on the right-hand side. The read command can accept muliple variable names.
Each variable will be assigned a word.
EXAMPLE
echo "What is your name?"
read name
read name1 name2 ...