EXPERIMENT NO. 4

UNIX FILE SYSTEM

AIM:

(A) Study of UNIX file system (tree structure).
(B) Study of .bashrc, /etc/bashrc and environment variables.
(C) Study File and Directory Permissions.

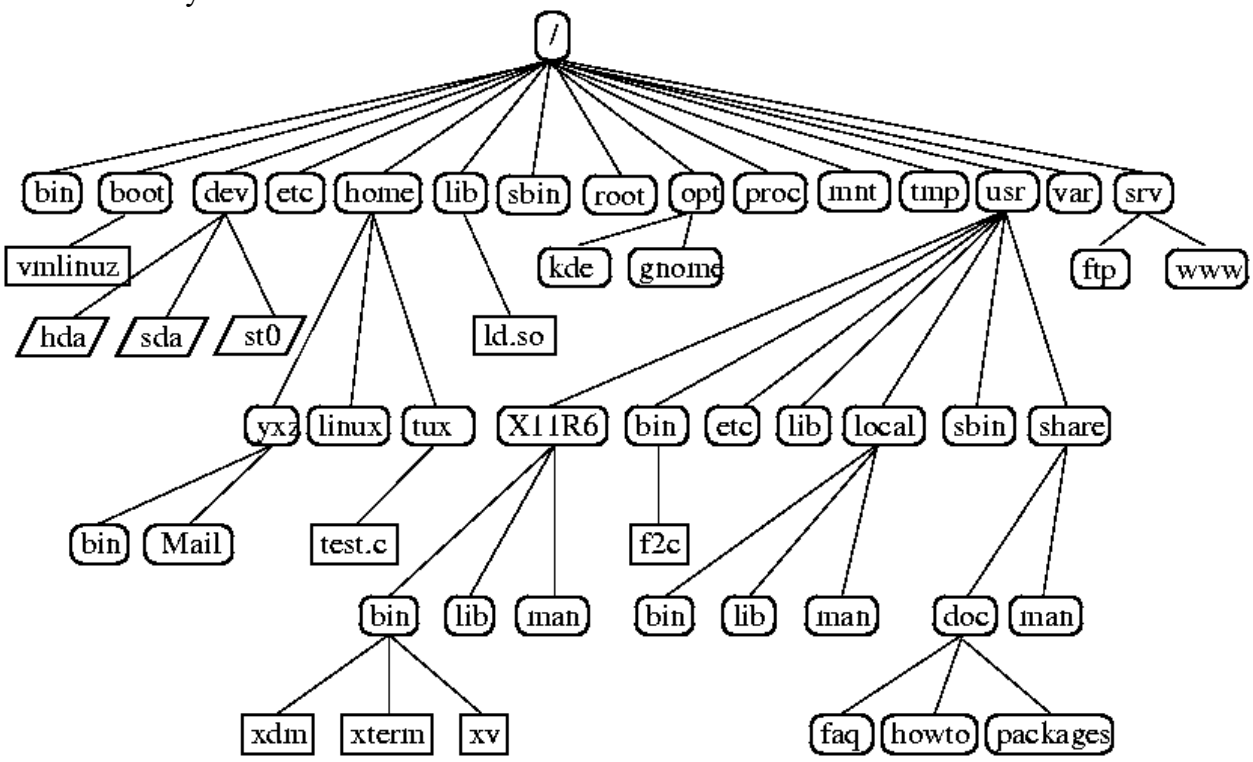
OBJECTIVE:

1. To introduce Basic Unix general purpose Commands.
2. To learn network Unix commands.
3. To learn file management and permission advance commands.

THEORY:

(A) Study of UNIX file system (tree structure).

Unix keeps track of files and directories of files using a file system. When you log in to your Unix account, you are placed in your "home" directory. Your home directory thus becomes your "present working directory" when you log in. In your home directory, you can create files and subdirectories. And in the subdirectories you create, you can create more subdirectories.
The commands that you issue at the Unix prompt relate to the files and folders and resources available from your present working directory. You certainly use and can refer to resources outside of your current working directory. To understand how this works, you need to know how the Unix file system is structured.



Directory or file	Description
/	The slash / character alone denotes the root of the filesystem tree.
/bin	Stands for *binaries* and contains certain fundamental utilities, such as ls or cp, that are needed to mount /usr, when that is a separate filesystem, or to run in one-user (administrative) mode when /usr cannot be mounted. In System V.4, this is a symlink to /usr/bin.
/boot	Contains all the files needed for successful booting process. In Research Unix, this was one file rather than a directory.[13]
/dev	Stands for *devices*. Contains file representations of peripheral devices and pseudo-devices. See also: Linux Assigned Names and Numbers Authority
/etc	Contains system-wide configuration files and system databases; the name stands for *et cetera*.[13] Originally also contained "dangerous maintenance utilities" such as init,[14] but these have typically been moved to /sbin or elsewhere.
/home	Contains user home directories on Linux and some other systems. In the original version of Unix, home directories were in /usr instead.[15] Some systems use or have used different locations still: macOS has home directories in /Users, older versions of BSD put them in /u, FreeBSD has /usr/home.
/lib	Originally *essential libraries*: C libraries, but not Fortran ones.[13] On modern systems, it contains the shared libraries needed by programs in /bin, and possibly loadable kernel module or device drivers. Linux distributions may have variants /lib32 and /lib64 for multi-architecture support.
/media	Default mount point for removable devices, such as USB sticks, media players, etc.
/mnt	Stands for *mount*. Empty directory commonly used by system administrators as a temporary mount point.
/opt	Contains locally installed software. Originated in System V, which has a package manager that installs software to this directory (one subdirectory per package).[16]
/proc	procfs virtual filesystem showing information about processes as files.
/root	The home directory for the superuser *root* - that is, the system administrator. This account's home directory is usually on the initial filesystem, and hence not in /home (which may be a mount point for another filesystem) in case specific maintenance needs to be performed, during which other filesystems are not available. Such a case could occur, for example, if a hard disk drive suffers physical failures and cannot be properly mounted.
/sbin	Stands for "system (or superuser) binaries" and contains fundamental utilities, such as init, usually needed to start, maintain and recover the system.
/srv	Server data (data for services provided by system).
/sys	In some Linux distributions, contains a sysfs virtual filesystem, containing

	information related to hardware and the operating system. On BSD systems, commonly a symlink to the kernel sources in /usr/src/sys.
/tmp	A place for temporary files not expected to survive a reboot. Many systems clear this directory upon startup or use tmpfs to implement it.
/unix	The Unix kernel in Research Unix and System V.[13] With the addition of virtual memory support to 3BSD, this got renamed /vmunix.
/usr	The "user file system": originally the directory holding user home directories,[15] but already by the Third Edition of Research Unix, ca. 1973, reused to split the operating system's programs over two disks (one of them a 256K fixed-head drive) so that basic commands would either appear in /bin or /usr/bin.[17] It now holds executables, libraries, and shared resources that are not system critical, like the X Window System, KDE, Perl, etc. In older Unix systems, user home directories might still appear in /usr alongside directories containing programs, although by 1984 this depended on *local customs*.[13]
/include	Stores the development headers used throughout the system. Header files are mostly used by the #include directive in C language, which historically is how the name of this directory was chosen.
/lib	Stores the needed libraries and data files for programs stored within /usr or elsewhere.
/libexec	Holds programs meant to be executed by other programs rather than by users directly. E.g., the Sendmail executable may be found in this directory.[18] Not present in the FHS until 2011;[19] Linux distributions have traditionally moved the contents of this directory into /usr/lib, where they also resided in 4.3BSD.
/local	Resembles /usr in structure, but its subdirectories are used for additions not part of the operating system distribution, such as custom programs or files from a BSD Ports collection. Usually has subdirectories such as /usr/local/lib or /usr/local/bin.
/share	Architecture-independent program data. On Linux and modern BSD derivatives, this directory has subdirectories such as man for manpages, that used to appear directly under /usr in older versions.
/var	Stands for *variable*. A place for files that may change often - especially in size, for example e-mail sent to users on the system, or process-ID lock files.
/log	Contains system log files.
/mail	The place where all incoming mails are stored. Users (other than root) can access their own mail only. Often, this directory is a symbolic link to /var/spool/mail.
/spool	Spool directory. Contains print jobs, mail spools and other queued tasks.
/tmp	The /var/tmp directory is a place for temporary files which should be preserved between system reboots.

(B) Study of .bashrc, /etc/bashrc and environment variables.

.bashrc is a shell script that Bash runs whenever it is started interactively. It initializes an interactive shell session. You can put any command in that file that you could type at the command prompt. BASH stands for Bourne Again Shell.
For example open a terminal window and enter the following command:
bash
Now within the same window enter this command:
bash
Every time you open a terminal window the bashrc file is performed.
The .bashrc file is a good place therefore to run commands that you want to run every single time you open a shell.
As an example open the .bashrc file using nano as follows:
nano ~/.bashrc
At the end of the file enter the following command:
echo "Hello $USER"
Save the file by pressing CTRL and O and then exit nano by pressing CTRL and X.
Within the terminal window run the following command:
bash
The word "Hello" should be displayed along with the username you are logged in as.

/etc/bashrc or /etc/bash.bashrc is the systemwide bash per-interactive-shell startup file. It is used system wide functions and aliases.

Environment Variables in Unix
Each process in Unix has its own set of environment variables. They're called environment variables because the default set of such variables consists mostly of session-wide variables used for configuration purposes.
Common environment variables in Unix:

- **USER** – username of a Unix user
- **HOME** – full path to a user's home directory
- **TERM** – terminal or terminal emulator used by a current user
- **PATH** – list of directories searched for executable files when you type a command in Unix shell
- **PWD** – current directory

(C) Study File and Directory Permissions.
File ownership is an important component of Unix that provides a secure method for storing files. Every file in Unix has the following attributes −

Owner permissions − The owner's permissions determine what actions the owner of the file can perform on the file.
Group permissions − The group's permissions determine what actions a user, who is a member of the group that a file belongs to, can perform on the file.
Other (world) permissions − The permissions for others indicate what action all other users can perform on the file.

The Permission Indicators
While using **ls -l** command, it displays various information related to file permission as follows –
$ls -l /home/amrood
-rwxr-xr-- 1 amrood users 1024 Nov 2 00:10 myfile
drwxr-xr--- 1 amrood users 1024 Nov 2 00:10 mydir

Here, the first column represents different access modes, i.e., the permission associated with a file or a directory.
The permissions are broken into groups of threes, and each position in the group denotes a specific permission, in this order: read (r), write (w), execute (x) −

- The first three characters (2-4) represent the permissions for the file's owner. For example, **-rwxr-xr--** represents that the owner has read (r), write (w) and execute (x) permission.
- The second group of three characters (5-7) consists of the permissions for the group to which the file belongs. For example, **-rwxr-xr--** represents that the group has read (r) and execute (x) permission, but no write permission.
- The last group of three characters (8-10) represents the permissions for everyone else. For example, **-rwxr-xr--** represents that there is **read (r)** only permission.

File Access Modes
The permissions of a file are the first line of defense in the security of a Unix system. The basic building blocks of Unix permissions are the **read**, **write**, and **execute** permissions, which have been described below −

- Read: Grants the capability to read, i.e., view the contents of the file.
- Write: Grants the capability to modify, or remove the content of the file.
- Execute: User with execute permissions can run a file as a program.

Directory Access Modes
Directory access modes are listed and organized in the same manner as any other file. There are a few differences that need to be mentioned −
Read Access to a directory means that the user can read the contents. The user can look at the **filenames** inside the directory.
Write Access means that the user can add or delete files from the directory.

Execute Executing a directory doesn't really make sense, so think of this as a traverse permission. A user must have **execute** access to the **bin** directory in order to execute the **ls** or the **cd** command.

Changing Permissions

To change the file or the directory permissions, you use the **chmod** (change mode) command. There are two ways to use chmod — the symbolic mode and the absolute mode.

Using chmod in Symbolic Mode

The easiest way for a beginner to modify file or directory permissions is to use the symbolic mode. With symbolic permissions you can add, delete, or specify the permission set you want by using the operators in the following table.

S.No.	Chmod operator & Description
1	+ Adds the designated permission(s) to a file or directory.
2	- Removes the designated permission(s) from a file or directory.
3	= Sets the designated permission(s).

Here's an example using **testfile**. Running **ls -1** on the testfile shows that the file's permissions are as follows −

```
$ls -l testfile
-rwxrwxr--  1 amrood   users 1024  Nov 2 00:10  testfile
```

Then each example **chmod** command from the preceding table is run on the testfile, followed by **ls –l**, so you can see the permission changes −

```
$chmod o+wx testfile
$ls -l testfile
-rwxrwxrwx  1 amrood   users 1024  Nov 2 00:10  testfile
$chmod u-x testfile
$ls -l testfile
-rw-rwxrwx  1 amrood   users 1024  Nov 2 00:10  testfile
$chmod g = rx testfile
```

```
$ls -l testfile
-rw-r-xrwx  1 amrood   users 1024  Nov 2 00:10  testfile
```

Here's how you can combine these commands on a single line −

```
$chmod o+wx,u-x,g = rx testfile
$ls -l testfile
-rw-r-xrwx  1 amrood   users 1024  Nov 2 00:10  testfile
```

Using chmod with Absolute Permissions

The second way to modify permissions with the chmod command is to use a number to specify each set of permissions for the file.

Each permission is assigned a value, as the following table shows, and the total of each set of permissions provides a number for that set.

Number	Octal Permission Representation	Ref
0	No permission	---
1	Execute permission	--x
2	Write permission	-w-
3	Execute and write permission: 1 (execute) + 2 (write) = 3	-wx
4	Read permission	r--
5	Read and execute permission: 4 (read) + 1 (execute) = 5	r-x
6	Read and write permission: 4 (read) + 2 (write) = 6	rw-
7	All permissions: 4 (read) + 2 (write) + 1 (execute) = 7	rwx

Here's an example using the testfile. Running **ls -1** on the testfile shows that the file's permissions are as follows −

```
$ls -l testfile
-rwxrwxr--  1 amrood   users 1024  Nov 2 00:10  testfile
```

Then each example **chmod** command from the preceding table is run on the testfile, followed by **ls –l**, so you can see the permission changes −

```
$ chmod 755 testfile
$ls -l testfile
-rwxr-xr-x  1 amrood   users 1024  Nov 2 00:10  testfile
$chmod 743 testfile
$ls -l testfile
-rwxr---wx  1 amrood   users 1024  Nov 2 00:10  testfile
$chmod 043 testfile
$ls -l testfile
----r---wx  1 amrood   users 1024  Nov 2 00:10  testfile
```

Changing Owners and Groups

While creating an account on Unix, it assigns a **owner ID** and a **group ID** to each user. All the permissions mentioned above are also assigned based on the Owner and the Groups.

Two commands are available to change the owner and the group of files −

- **chown** − The **chown** command stands for **"change owner"** and is used to change the owner of a file.
- **chgrp** − The **chgrp** command stands for **"change group"** and is used to change the group of a file.

Changing Ownership

The **chown** command changes the ownership of a file. The basic syntax is as follows −

```
$ chown user filelist
```

The value of the user can be either the **name of a user** on the system or the **user id (uid)** of a user on the system.

The following example will help you understand the concept −

```
$ chown amrood testfile
```

```
$
```

Changes the owner of the given file to the user **amrood**.

NOTE − The super user, root, has the unrestricted capability to change the ownership of any file but normal users can change the ownership of only those files that they own.

Changing Group Ownership

The **chgrp** command changes the group ownership of a file. The basic syntax is as follows −

```
$ chgrp group filelist
```

The value of group can be the **name of a group** on the system or **the group ID (GID)** of a group on the system.

Following example helps you understand the concept −

```
$ chgrp special testfile
$
```

Changes the group of the given file to **special** group.

OUTCOME:

1. Identified the basic Unix general purpose commands.
2. Studied how to change the ownership and file permissions using advance Unix commands.
3. Applied basic commands for administrative task.

CONCLUSION: We have successfully studied UNIX File System.