EXPERIMENT 2

Linux commands for redirection, pipes, filters, job control, file ownership, file permissions, links and file system hierarchy.

1. Redirecting :

- Mostly all command gives output on screen or take input from keyboard, but in Linux it is possible to send output to file or to read input from file.
- For e.g.

```
$ ls
```

- This 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
```

2. Pipes :

- The -r option to sort, sorts the data in reverse-alphabetical order. If you want to list the files in your current directory in reverse order, one way to do it is follows:

```
$ ls > file_list
$ sort –r file-list
```

- Here, you save the output of ls in a file, and then run sort -r on that file. But this is unwieldy and uses a temporary file to save the data from ls.The solution is pipelining.
- **Pipelining:** This is a shell feature that connects a string of commands via a pipe. The **`stdout`** of the first command is sent to the **`stdin`** of the second command. In this case, send the **`stdout`** of **`ls`** to the **`stdin`** of **`sort`**. Use the ``|" symbol to create a pipe, as follows:

```
ls | sort –r
```

- You can pipe more than two commands together. The command `head` is a filter that displays the first lines from an input stream (in this case, input from a pipe).

```
$ ls | sort –r | head -1
```

- where `head -1` displays the first line of input that it receives (in this case, the stream of reverse-sorted data from `ls`).

3. Filters

- If a Linux command accepts its input data from the standard input and produces its output (result) on standard output is known as a **filter**.
- Filters usually works with Linux pipes.
- The syntax is:

```
command1 | command2
command1 file.txt | command2
command1 args < input.txt | command2
```

- Where, command2 is a filter command.
- Consider the following example:

```
sort < sname | uniq > u_sname
```

- The uniq command is filter, which takes its input from the sort command and passes output as input to uniq command; Then uniq command output is redirected to "u_sname" file.

4. Job Control

- When you execute a unix shell-script or command that takes a long time, you can run it as a background job.

Executing a background job

- Appending an ampersand (&) to the command runs the job in the background.
- Following example finds all the files under root file system that changed within the last 24 hours.

```
# find / -ctime -1 > /tmp/changed-file-list.txt &
```

Sending the current foreground job to the background using CTRL-Z and bg command

- You can send an already running foreground job to background as explained below:
    - Press ‘CTRL+Z’ which will suspend the current foreground job.

- Execute bg to make that command to execute in background.

- For example, if you've forgot to execute a job in a background, you don't need to kill the current job and start a new background job. Instead, suspend the current job and put it in the background as shown below.

```
# find / -ctime -1 > /tmp/changed-file-list.txt

# [CTRL-Z]
[2]+  Stopped                 find / -ctime -1 > /tmp/changed-file-list.txt

# bg
```

View all the background jobs using jobs command

- You can list out the background jobs with the command **jobs**. Sample output of jobs command is

```
# jobs
[1]   Running                 bash download-file.sh &
[2]-  Running                 evolution &
[3]+  Done                    nautilus .
```

Taking a job from the background to the foreground using fg command

- You can bring a background job to the foreground using **fg command**. When executed with-out arguments, it will take the most recent background job to the foreground.

```
# fg
```

- If you have multiple background ground jobs, and would want to bring a certain job to the foreground, execute jobs command which will show the job id and command.
- In the following example, fg %1 will bring the job#1 (i.e download-file.sh) to the foreground.

```
# jobs
[1]   Running                 bash download-file.sh &
[2]-  Running                 evolution &
[3]+  Done                    nautilus .

# fg %1
```

Kill a specific background job using kill %

- If you want to kill a specific background job use, kill %job-number. For example, to kill the job 2 use.

```
# kill %2
```

5. File ownership

- You can use ls -l command (list information about the FILEs) to find out the file / directory owner and group names

```
$ ls -l filename
```

- You can use **chown** and **chgrp** commands to change the owner or the group of a particular file or directory.

```
 $ chown root tmpfile
```

  - Example:

```
 ls -l tmpfile
-rw-r--r-- 1 himanshu family 0 2012-05-22 20:03 tmpfile

# chown root tmpfile

# ls -l tmpfile
-rw-r--r-- 1 root family 0 2012-05-22 20:03 tmpfile
```

  - Now the owner of the file was changed from ‘himanshu’ to ‘root’.
  - Example:

```
# ls -l tmpfile
-rw-r--r-- 1 himanshu family 0 2012-05-22 20:03 tmpfile

# chown :friends tmpfile

# ls -l tmpfile
-rw-r--r-- 1 himanshu friends 0 2012-05-22 20:03 tmpfile
```

  - The group of the file changed from ‘family’ to ‘friends’.

- Using the syntax '<newOwner> **:** <newGroup>', the owner as well as group can be changed in one go.
- Example :

```
# chown himanshu:friends tmpfile
```

-

.

6. File Permissions

- Unix file and directory permission is in the form of a 3×3 structure. i.e Three permissions (read, write and execute) available for three types of users (owner, groups and others).
- Three file permissions:
  - **read**: permitted to read the contents of file.
  - **write**: permitted to write to the file.
  - **execute**: permitted to execute the file as a program/script.
- Three directory permissions:
  - **read**: permitted to read the contents of directory (view files and sub-directories in that directory).
  - **write**: permitted to write in to the directory. (create files and sub-directories in that directory)
  - **execute**: permitted to enter into that directory.
- Numeric values for the read, write and execute permissions:
  - read 4
  - write 2
  - execute 1
- Three types of users
  - **u** - Owner
  - **g** - Group
  - **o** or **a** - All Users
- In the output of ls -l command, the 9 characters from 2nd to 10th position represents the permissions for the 3 types of users. For example,

```
-rw-r--r--  1 sathiya sathiya  272 Mar 17 08:22 test.txt
```

- In the above example:
  - User (sathiya) has read and write permission
  - Group has read permission
  - Others have read permission

- Changing permissions.

- The command `chmod` is used to set the permissions on a file. Only the owner of a file may change the permissions on that file.
- The syntax of `chmod` is
  `chmod {a,u,g,o}{+,-}{r,w,x} filenames`
- Briefly, you supply one or more of **a**ll, **u**ser, **g**roup, or **o**ther. Then you specify whether you are adding rights (+) or taking them away (-). Finally, you specify one or more of **r**ead, **w**rite, and e**x**ecute.
-

7. File system hierarchy

Files systems break files down into two logical categories:

- Shareable vs. unsharable files
- Variable vs. static files

Shareable files are those that can be accessed locally and by remote hosts; *unsharable* files are only available locally. *Variable* files, such as documents, can be changed at any time; *static* files, such as binaries, do not change without an action from the system administrator.

File System Hierarchy Standard (FHS) organisation

1) The `/boot/` Directory

- The `/boot/` directory contains static files required to boot the system, such as the Linux kernel. These files are essential for the system to boot properly.

2) The /dev/ Directory

- The /dev/ directory contains file system entries which represent devices that are attached to the system. These files are essential for the system to function properly.

3.) The /etc/ Directory

- The /etc/ directory is reserved for configuration files that are local to the machine. No binaries are to be put in /etc/. Any binaries that were once located in /etc/ should be placed into /sbin/ or /bin/.
- The X11/ and skel/ directories are subdirectories of the /etc/ directory.
- The /etc/X11/ directory is for X Window System configuration files such as XF86Config.
- The /etc/skel/ directory is for "skeleton" user files, which are used to populate a home directory when a user is first created.

4.) The /lib/ Directory

- The /lib/ directory should contain only those libraries needed to execute the binaries in /bin/ and /sbin/.
- These shared library images are particularly important for booting the system and executing commands within the root file system.

5.) The /mnt/ Directory

- The /mnt/ directory is for temporarily mounted file systems, such as CD-ROMs and 3.5 diskettes.

6.) The /opt/ Directory

- The /opt/ directory provides storage for large, static application software packages.
- A package placing files in the /opt/ directory creates a directory bearing the same name as the package. This directory, in turn, holds files that otherwise would be scattered throughout the file system, giving the system administrator an easy way to determine the role of each file within a particular package.

- For example, if sample is the name of a particular software package located within the /opt/ directory, then all of its files are placed in directories inside the /opt/sample/ directory, such as /opt/sample/bin/ for binaries and /opt/sample/man/ for manual pages.
- Large packages that encompass many different sub-packages, each of which accomplish a particular task, are also located in the /opt/ directory, giving that large package a way to organize itself. In this way, our sample package may have different tools that each go in their own sub-directories, such as /opt/sample/tool1/ and /opt/sample/tool2/, each of which can have their own bin/, man/, and other similar directories.

7.) The `/proc/` Directory

- The `/proc/` directory contains special files that either extract information from or send information to the kernel.
- Due to the great variety of data available within `/proc/` and the many ways this directory can be used to communicate with the kernel, an entire chapter has been devoted to the subject.

8.) The `/sbin/` Directory

- The `/sbin/` directory stores executables used by the root user. The executables in `/sbin/` are only used at boot time and perform system recovery operations. Of this directory, the FHS says:
- `/sbin` contains binaries essential for booting, restoring, recovering, and/or repairing the system in addition to the binaries in `/bin`. Programs executed after `/usr/` is known to be mounted (when there are no problems) are generally placed into `/usr/sbin`. Locally-installed system administration programs should be placed into `/usr/local/sbin`.

9.) The `/usr/` Directory

- The `/usr/` directory is for files that can be shared across multiple machines. The `/usr/` directory is often on its own partition and is mounted read-only. At minimum, the following directories should be subdirectories of `/usr/`:

```
/usr
      |- bin/
      |- dict/
      |- doc/
      |- etc/
      |- games/
      |- include/
      |- kerberos/
```

```
|- lib/
|- libexec/
|- local/
|- sbin/
|- share/
|- src/
|- tmp -> ../var/tmp/
|- X11R6/
```

10.) The `/usr/local/` Directory

- The `/usr/local` hierarchy is for use by the system administrator when installing software locally. It needs to be safe from being overwritten when the system software is updated. It may be used for programs and data that are shareable among a group of hosts, but not found in `/usr`.
- The `/usr/local/` directory is similar in structure to the `/usr/` directory. It has the following subdirectories, which are similar in purpose to those in the `/usr/` directory:

```
/usr/local
       |- bin/
       |- doc/
       |- etc/
       |- games/
       |- include/
       |- lib/
       |- libexec/
       |- sbin/
       |- share/
       |- src/
```

- For instance, if the `/usr/` directory is mounted as a read-only NFS share from a remote host, it is still possible to install a package or program under the `/usr/local/` directory.

11.) The `/var/` Directory

- Since the FHS requires Linux to mount `/usr/` as read-only, any programs that write log files or need `spool/` or `lock/` directories should write them to the `/var/` directory.
- `/var/` is for variable data files. This includes spool directories and files, administrative and logging data, and transient and temporary files.
- Below are some of the directories found within the `/var/` directory:

```
/var
  |- account/
  |- arpwatch/
  |- cache/
  |- crash/
```

```
|- db/
|- empty/
|- ftp/
|- gdm/
|- kerberos/
|- lib/
|- local/
|- lock/
|- log/
|- mail -> spool/mail/
|- mailman/
|- named/
|- nis/
|- opt/
|- preserve/
|- run/
+- spool/
     |- at/
     |- clientmqueue/
     |- cron/
     |- cups/
     |- lpd/
     |- mail/
     |- mqueue/
     |- news/
     |- postfix/
     |- repackage/
     |- rwho/
     |- samba/
     |- squid/
     |- squirrelmail/
     |- up2date/
     |- uucppublic/
     |- vbox/
|- tmp/
|- tux/
|- www/
|- yp/
```

8. Links

1. Soft Link

- Linux OS recognizes the data part of this special file as a reference to another file path. The data in the original file can be accessed through the special file, which is called as Soft Link.
- To create a soft link, do the following (ln command with -s option):

```
$ ln -s /full/path/of/original/file /full/path/of/soft/link/file
```

2. Hard Link

- With Hard Link, more than one file name reference the same inode number. Once you create a directory, you would see the hidden directories "." and ".." . In this, "." directory is hard linked to the current directory and the ".." is hard linked to the parent directory.
- When you use link files, it helps us to reduce the disk space by having single copy of the original file and ease the administration tasks as the modification in original file reflects in other places.
- To create a hard link, do the following (ln command with no option):

```
$ ln /full/path/of/original/file /full/path/of/hard/link/file
```

3. Symbolic Link

- Symbolic links, or **symlinks**, are another type of link, which are different from hard links. A symbolic link lets you give a file another name, but doesn't link the file by inode.
- The command `ln -s` creates a symbolic link to a file.
- For example, if you use the command

```
$ ln -s foo bar
```

- It will create a symbolic link named `bar` that points to the file `foo`. If you use `ls -i`, you'll see that the two files have different inodes, indeed.
- `$ ls -i foo bar`
- However, using ls -l, we see that the file `bar` is a symlink pointing to `foo`.
- `$ ls -l foo bar`