Lecture – 14

SECTION -C

Getting Started with UNIX

Introduction

- UNIX files & Directories
- Basic operations on Files
- File Permissions

Already covered following topics in Unit -4 i.e.

Theoretical Concepts of UNIX O.S.

UNIX Files & Directories

- **The file**

✓ Ordinary files, Device Files, Directory Files.. **We'll discuss various commands later in this chapter… but that will also be a part of this topic.**

- **Directories:**

✓ Directory Structure(the Parent child Relationship-UNIX file system)

✓ Checking your Directory

✓ pwd : Checking your current directory

✓ cd : Changing Directory

✓ mkdir : making directories

✓ rmdir: removing directories

✓ Current Directory

Basic Operation on Files:

1. **cat : DISPLAYING AND CREATING FILES:**

   cat is one of the most well-known commands of the UNIX system. It is mainly used to display the contents of a small file on the terminal:

```
$ cat dept.lst
01|accounts|6213
02|progs |5423
03| marketing | 6521
04|sales | 1006
```

cat, like several other UNIX commands, also accepts more than one filename as arguments:

```
$ cat  chap01  chap02
```

Contents of the second file are shown immediately after the first file without any header information. In other words, cat con**cat**enates the two files – hence its name.

2. Using cat to Create a File:

cat is also useful for creating a file. Enter the command cat followed by > (the right chevron) character and the filename (for eg. *file1*):

```
$ cat > file1
```

A > symbol following the command means that the output goes to the filename following it.

```
[Ctrl – d]
$ _
```

when the command line is terminated with *[Enter]* , the prompt vanishes . **cat** now waits to take input from the user. Enter the three lines, each followed *[Enter].* Finally press *[Ctrl – d]* to signify the end of input to the stream.

3. cp : COPYING A FILE

The **cp** command copies a file or a group of files. It creates an exact image of the file on disk with a different name. The syntax requires at least two filenames to be specified in the command line. When both are ordinary files, the first is copied to the second:

```
$ cp  chap01  unit1
```

If the destination file (unit1) doesn't exist, it will first be created before copying takes place. If not, it will simply be overwritten without any warning from the system. So be careful when you choose your destination filename. Just check with the **ls** command whether or not the file exists.

If there is only file to be copied, the destination can be either an ordinary or directory. You then have the option of choosing your destination filename. The following example shows two ways of copying a file to the progs directory:

```
$ cp chap01 progs/unit1      chap01 copied to unit1 under progs.
$ cp chap01 progs            chap01 retains its name under progs
```

4. **rm: DELETING FILES:**

- The rm command deletes one or more files. It normally operate silently and should be used with caution. The following command deletes three files :

  $ rm chap01 chap02 chap03

- A file once deleted can't be recovered. rm won't normally remove a directory but it can remove files from one.

  You can remove two chapters from the progs directory without having to "cd" to it:

  $ rm progs/chap01 progs/chap02 **Or** **$ rm progs chap0[12]**

- You may sometimes need to delete all files in a directory as a part of clean up operation. The * , when used by itself, represents all files, and you can then use "rm" like this:

5. **mv : RENAMING FILES**

The mv command renames (move files). It has two distinct functions:

- ✓ It renames a file(or directory)
- ✓ It moves a group of files to a different directory.

mv doesn’t create a copy of the file; it merely renames it. No additional space is consumed on disk during renaming.

To rename the file chap01 to man01, you should use:

```
$  mv    chap01   man01
```

If destination file doesn’t exist, it will be created. For the above example, mv simply replaces the filename in the existing directory entry with the new name.

It can also be used to rename a directory, for instance pis to perdir:

```
$  mv   pis  perdir
```

File Permissions

- UNIX has a simple and well-defined system of assigning permissions to files. Lets issue the ls – l command once again to view the permissions of a few files:

```
$ ls –l chap02 dept.lst dateval.sh
-rwxr-xr- -    1     kumar            metal    20500 May 10 19:21 chap 02
-rwxr-xr-      1     kumar            metal    890    Jan  29  23:17 dateval.sh
-rw-rw-rw-   1      kumar              metal     84       Feb 12  12:30  dept.lst
```

- Observe that the first column that represents the file permissions. These permissions are also different for the three files.
- UNIX follows a three-tiered file protection system that determines a file's access rights.
- To understand how this system works, lets break up the permission string of the file chap-2 into three groups. The initial - (in the first column) represents an ordinary file and is left out of the permissions string:

```
r w x            r – x              r - -
```

- Each group here represents a **category**, that contains three slots, representing the read, write and execute permissions of the file – in that order.

✓ r indicates read permission, which means cat can display the files.

✓ w indicates write permission, you can edit such a file with an editor.

✓ x indicates execute permission; the file can be executed as a program.

✓ The – shows the absence of the corresponding permission.

- The first group (rwx) has all three permissions. The file is readable, writable and executable by the owner of the file, kumar.
- The third column shows kumar as the owner and first permissions group applies to kumar. You have to login with the username kumar for these privileges to apply to you.
- The second (r-x) has a hyphen in the middle slot, which indicates the absence of write permission by the group owner of the file. This group owner is **metal,** and all users belonging to the metal group have read and

- The third group (r- -) has the write and execute bits absent. This set of permissions is applicable to others, i.e. those who are neither the owner nor belong to the metal group. This category (others) is often referred to as the world. This file is not world-writable.
- You can set different permissions for the three categories of users – owner, group and others.

Applications in Games

There are lots of fun things and games you can use in UNIX. Most of the ones listed below are local to Brown University. Try each of these commands. Check the man pages or the links below if you have trouble, but note, some of the commands do not have man pages. Have fun!

- **banner**
- **figlet**
- **WhatsForDinner**
- **food dilbert**
- **Forecast**
- **fortune say**
- **Xteddy**
- **xdeady**
- **BattleTris**
- **nethack**
- **Netris**
- **xbill**
- **xblast**
- **xboing**
- **xroach**

Research

- **Unix Commands** CCR's computing resources are primarily Linux based and therefore using them requires a basic understanding of the Unix operating system. Some basic commands are provided below.
- **Basic Unix Commands** CCR Reference Card for Linux/UNIX commands pdf
- Show pathname of current directory: pwd
- List files: ls
- Make a directory: mkdir directory-name
- Change directory: cd directory-name
  - Change directory back to home directory: cd
- Copy a file: cp old-filename new-filename
- View a file:
  - cat filename
  - more filename
  - less filename
- Edit a file:
  - emacs filename
  - vi filename
- Delete a file: rm filename
  - Delete a directory (recursively): rm -R directory-name
    - All files and subdirectories are deleted
- Move a file: mv old-filename new-filename
- Change permissions:
  - Arguments to chmod command: ugo+-rwx
    - where ugo are user, group and other; rwx are read, write and execute
  - Add execute permission for yourself: chmod u+x filename
  - Remove read, write and execute for group and other from a directory its contents: chmod -R go-rwx directory-name