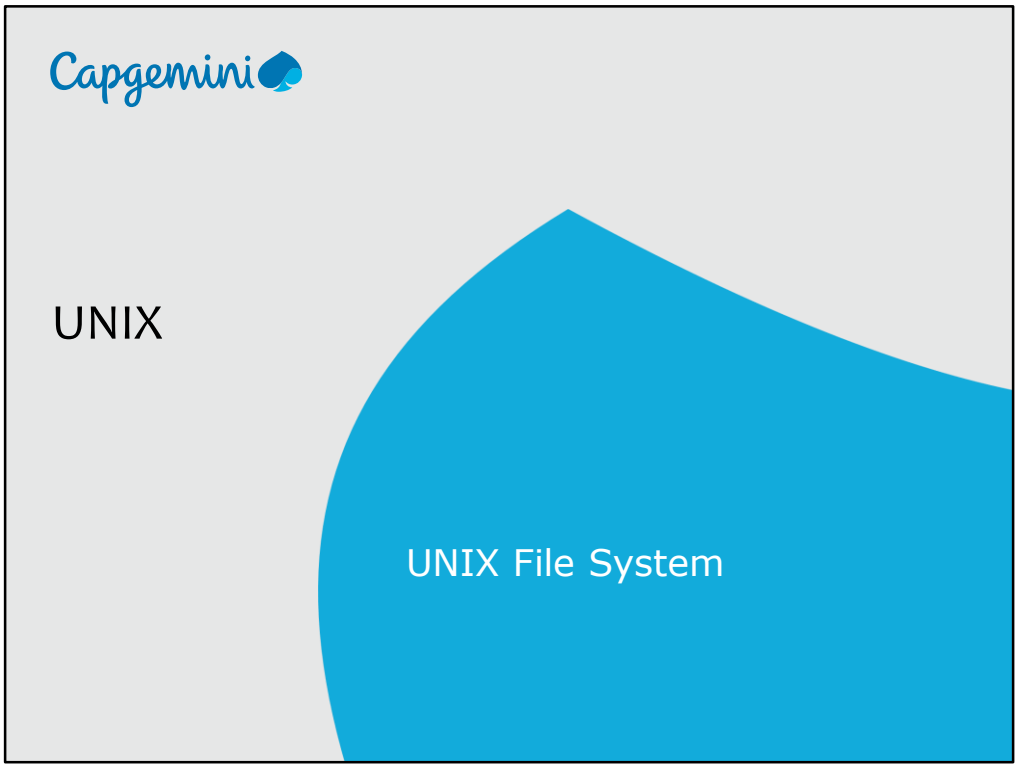
Instructor Notes:



Instructor Notes:

Lesson Objectives

In this lesson, you will learn:

- UNIX File system
- File types
- File permissions
- Commands related to file permission
  - mkdir, cd, cat etc…

Instructor Notes:

2.1: File System

Overview

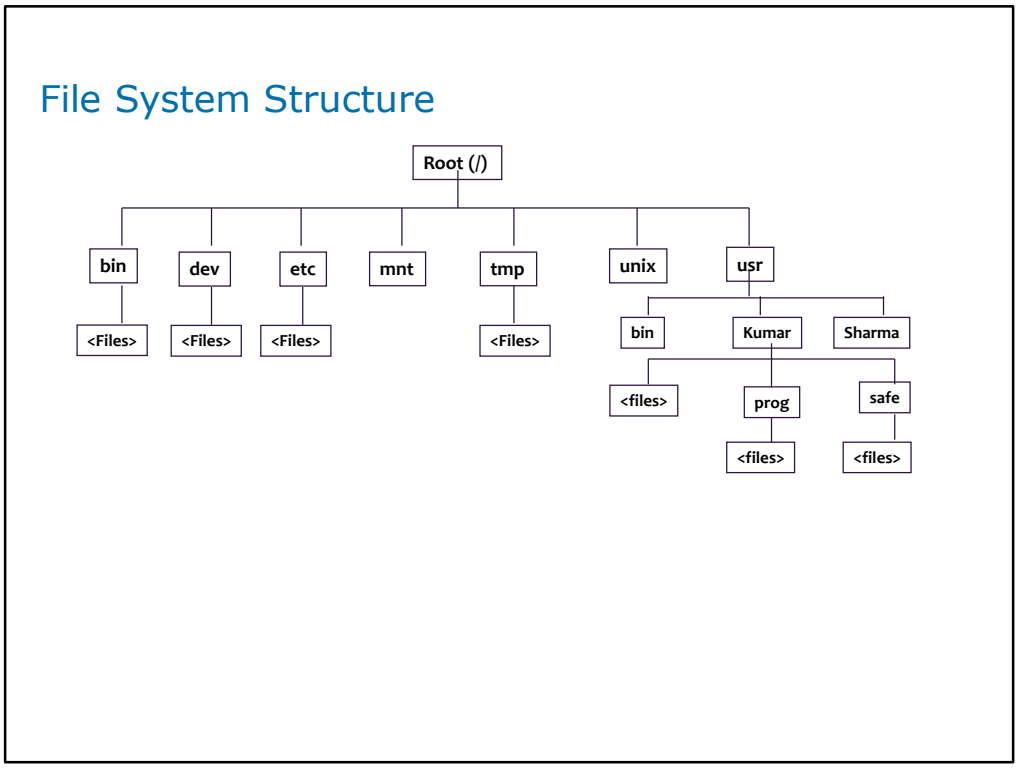
Let us discuss a File System with respect to the following:

- Hierarchical Structure
- Consistent Treatment of Data: Lack of file format
- The Treatment of Peripheral Devices as Files
- Protection of File Data

The UNIX File System:

UNIX works with a large number of files, which can belong to several different users. It becomes imperative for UNIX to organize files in a systematic fashion. The simple file system of UNIX is designed as an elaborate **Storage System** with separate compartment becoming available to store files. The system is widely adopted by different systems including DOS.

Instructor Notes:



File System Structure:

The file system in UNIX is hierarchical. The **root**, a directory file represented by /, is at the top of the hierarchy and has several subdirectories (branches) under it. An example of a typical UNIX file structure is given in the above slide.

There can be more than one file system, each with its own root, in a single machine. The number of file systems cannot be less than the number of physical disks but can certainly exceed the latter. However, a file system cannot span multiple disks. If there are more than one file systems, there will always be one main file system.

Instructor Notes:

File System Structure

/ bin : commonly used UNIX Commands like who, ls

/usr/bin : cat, wc etc. are stored here

/dev : contains device files of all hardware devices

/etc : contains those utilities mostly used by system administrator

- Example: passwd, chmod, chown

Instructor Notes:

File System

/tmp : used by some UNIX utilities especially vi and by user to store temporary files

/usr : contains all the files created by user, including login directory

/unix : kernel

Release V:

- It does not contain / bin.
- It contains / home instead of /usr.

Instructor Notes:

2.2: File Types

File Types in UNIX

We have the following file types in UNIX:

- Regular File
- Directory File
- Device File

UNIX Files:

A UNIX file consists of a sequence of characters, and there are no restrictions on the file structure. The file consists only the actual bytes – and no extra information like its own size, attributes, or even an end of file marker. Extra information is stored in a structure called as **inode (index node)**.

In UNIX, for every file, an inode is stored irrespective of its type.
Everything in UNIX is treated as a file, may be it is a user created or system file, a directory or a peripheral device. UNIX treats all the files in a consistent format. The lack of file formats and the consequent uniformity of files is of advantage to the programmer, as programmers do not need to worry about the file types. Further most of the standard programs work with any file.

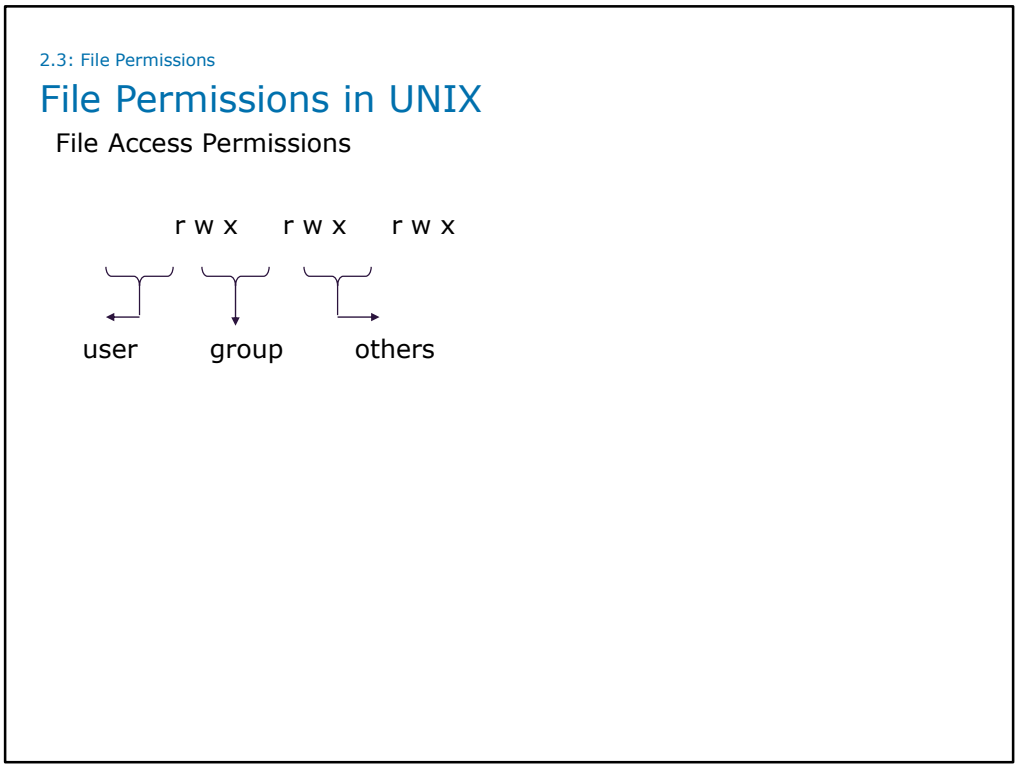
Though everything is treated as a file, the significance of the file attributes is dependent on the categorization of files as Ordinary files, Directory Files and Device files.

Ordinary Files or Regular File :

These are also called as regular files. This is the "traditional" file, which can contain programs, data, object, and executable code, as well as all the UNIX programs and other user created files.

All text files belong to this category.

Instructor Notes:



File Permissions:

Every file in UNIX has a set of permissions using which it is determined "who can do what" with the file. This is stored as part of **inode** information, and is useful for maintaining file security.

There are three categories of users: **Owner** (the user), **Group** (user is a member of which group), and **Others** (everybody else on the system). Access permissions of read (examining contents), write (changing contents) and execute (running program) are assigned to a file. These permissions are assigned to file owner, group and others.

It is the file permissions based on which reading, writing or executing a file is decided for a particular user. It helps in giving restricted access to a file. The commands to assign permissions and also to change the same are discussed later.

Instructor Notes:

File Permissions in UNIX

Permissions are associated with every file, and are useful for security.

There are three categories of users:

- Owner (u)
- Group (g)
- Others (o)

There are three types of "access permissions":

- Read (r)
- Write (w)
- Execute (e)

Instructor Notes:

2.4: File Related Commands

pwd Command

The pwd command checks current directory.

```
$ pwd
```

- **Output:** /usr/Kumar

pwd Command:

The **pwd** command is used to print the name of the current working directory.

```
$ pwd
```

Output: /usr1/deshpavn

Instructor Notes:

cd Command

The cd command changes directories to specified directory

The directory name can be specified by using absolute path (Full Path) or relative path

```
$ pwd
```

- **Output:** /usr/kumar

```
$ cd Prog
$ pwd
```

- **Output:** /usr/kumar/Prog

cd (change directory) Command:
The cd command is used to change the current directory to the directory specified as the argument to the command. The argument can contain absolute as well as relative paths.

```
$ cd deshpavn
$ pwd
```

Output: /usr1/deshpavn

Absolute Path and Relative Path
To locate the file or directory which is not in current working directory full path can be given.
If the specified path starts from root directory i.e from '/' the it is called as **absolute path or full path**. Otherwise it is called as **relative path** because the given path is relative to the current working directory

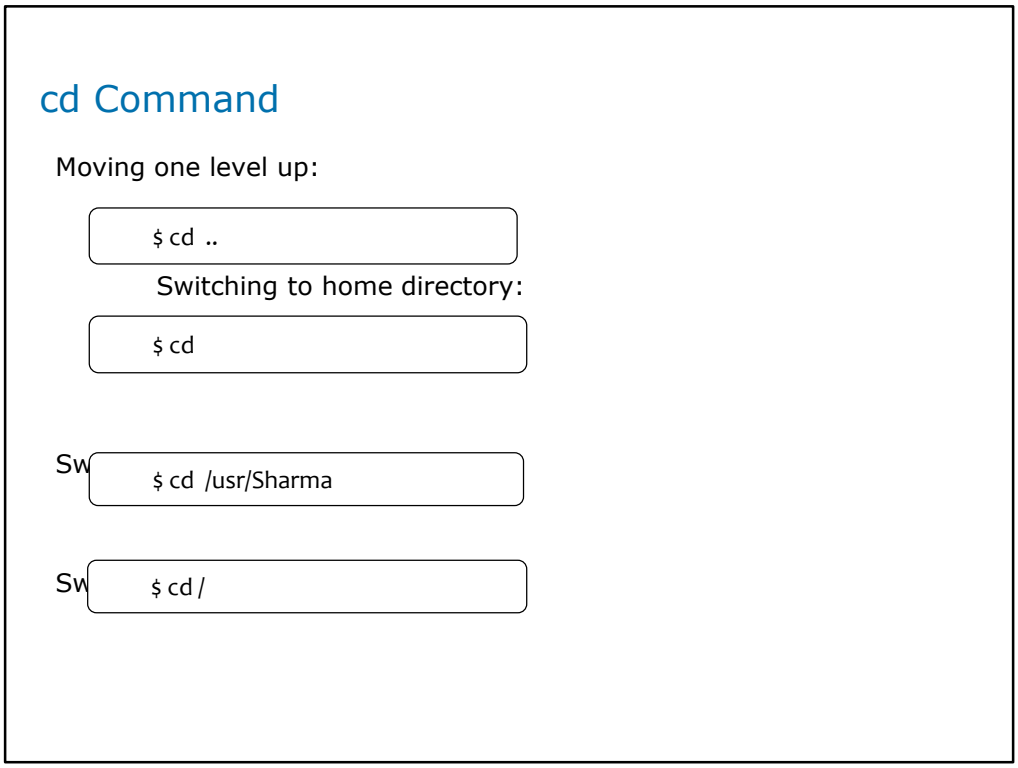
e.g If current working directory is /usr, then to change the directory to prog
Use following command
$pwd
/usr
$cd **kumar/prog**

In the above example, specified path to prog directory is not starting with '/'. It is relative path from /usr directory(current working directory).

The same command can be given as
$cd /usr/kumar/prog

In above example, since the given path is starting from root directory i.e '/' hence this is called as absolute path or full path.

Instructor Notes:



cd (change directory) Command:
To move one level up in the file system, the command is used as follows:

```
$ pwd
```

Output: /usr1/deshpavn

```
$ cd ..
$ pwd
```

Output: /usr1

When the command is used without any parameters, it switches to home directory as the current directory.

Output: /usr1

```
$ pwd
```

Output: /usr1/deshpavn

```
$ cd
$ pwd
```

Instructor Notes:

logname Command

The logname command checks the login directory.

```
$ logname
```

Output: Kumar

logname Command - Checking Login Directory :

The **logname** command can be used to display the name of login directory, irrespective of what is the current working directory.

```
$ logname
```

Output: deshpavn

Instructor Notes:

ls Command

The ls command lists the directory contents.

Example:

```
$ ls
```

Output:

a.out
chap1
chap2
test
test.c

ls Command – Listing directory contents:

The **ls** command is used to list all the contents of the specified directory (in case no argument is specified, current directory is assumed).

```
$ ls
```

Output:

file1.txt
file2.txt
file3.txt
Mail

```
$ ls
```

Output:

newfile.txt

Instructor Notes:

ls Command

Options available in ls command:

Option	Description
-x	**Displays multi columnar output (prior to Release 4)**
-F	**Marks executables with *and directories with /**
-r	**Sorts files in reverse order (ASCII collating sequence by default)**
-l	**The long listing showing seven attributes of a file**
-d	**Forces listing of a directory**
-a	**Shows all files including ., .. And those beginning with a dot**

ls Command – Listing directory contents (contd.):
The ls command comes with several options, which are listed in the table. Many of these options can be combined to get relevant output. The command can also be used with more than one file name that is specified.

Instructor Notes:

ls Command

Options available in ls command:

Option	Description
-t	**Sorts files by modification time**
-R	**Recursive listing of all files in sub-directories**
-u	**Sorts files by access time (when used with the –t option)**
-i	**Shows i-node number of a file**
-s	**Displays number of blocks used by a file**

ls Command – Listing directory contents (contd.):
Some of the examples are considered here:
Example 1: To produce multiple column output (-x):

```
$ ls -x
```

Output: file1.txt file2.txt file3.txt mail

Example 2: To identify directories and executable files (-F):
Here two tags, that is * and /, are used to indicate executable file and a directory respectively.

Instructor Notes:

ls Command

Example:

```
$ ls – l
```

- It displays output as follows which includes 7 columns
  total 8:

```
-rw-rw-rw-        1 Kumar group 44   May 9 09:08 dept.h
-rw-rw-rw-        1 Kumar group 212 May 9 09:08 dept.q
-rw-rw-rw-        1 Kumar group 154 May 9 09:08 emp.h
```

ls Command – Listing directory contents (contd.):
Example 3: To get an informative listing i.e. long listing (-l)

```
$ ls -l
```

```
total 12
-rw-r--r--  1 deshpavn group      60 Mar 29 10:43 file1.txt
-rw-r--r--  1 deshpavn group      61 Mar 29 10:44 file2.txt
-rw-r--r--  1 deshpavn group      60 Mar 29 10:44 file3.txt
drwx------  2 deshpavn group     512 Mar 14 10:00 mail
drwxr-xr-x  2 deshpavn group     512 Mar 29 10:46 testdir1
drwxr-xr-x  2 deshpavn group     512 Mar 29 10:47 testdir2
```

Seven attributes are listed which are explained in further slides in order as they appear in listing.

Type and Permissions associated with file:

The first bit "**d**" indicates that file is a **directory file**, while "**–**" indicates that file is a **regular file**. In case of device files, one would normally find bit "**c**" for **character based device** and bit "**b**" for **block based devices**.

File permissions are the read, write, and execute permissions set for owner, group, and others. Significance of these permissions with respect to directories are discussed in another example.

Instructor Notes:

ls Command

Consider the first column:

Field1 --> mode

- r w x r w x r w x

  - --> user permissions
  - --> group permissions
  - --> others permissions

ls Command – Listing directory contents (contd.):

Permissions are interpreted as follows:

For a directory: The **"r" permission** implies that one can find out the contents of the directory using commands like ls. The **"w" permission** implies that it is possible to create and delete files in this directory. It is possible to remove even the files that are write-protected. The **"x" permission** means search rather than execute. That is, it implies that the directory can be searched for a file. Also, when 'x' permission is set for a directory, one can do change dir to that directory and not otherwise.

Example: Using "r- -" will ensure that users can see the directory contents using ls, but cannot really use the contents.

To list all hidden files (-a):

The ls command does not list all the hidden files unless –a attribute is used.

To display inode information and number of blocks used for file:

```
$ ls -i
```

658 file1.txt
3952 file2.txt
3956 file3.txt
434 mail
3957 testdir1
3960 testdir2

```
$ ls -i testdir1
```

658 fileln1.txt

Instructor Notes:

ls Command

File type

- 1 st character represents file type:

  - r w x r w x r w x
    - - --> regular file
    - d --> directory file
    - c --> character - read
    - b --> block read

Instructor Notes:

ls Command

Field2 : indicates number of links
Field3 : File owner id
Field4 : Group id
Field5 : File size in bytes
Field6 : Date/time last altered
Field7 : Filename

Instructor Notes:

cat Command

The cat command is used for displaying and creating files.

- To display file:

```
$ cat dept.lst
```

```
01|accounts|6213
02|admin|5423
:
06|training|1006
```

- To create a file:

```
$cat > myfile
```

  - This is a new file
  - Press ctrl-d to save the contents in file myfile

cat Command (Displaying and Creating files):

The **cat** command can be used to display one or more files (if there is more than one file, then contents of the remaining immediately follow without any header information). To control the scrolling of text on screen, you can use **<Control-s>** and **<Ctrl-q>**.

```
$ cat file1.txt
```

Output: This is a sample text file.

This is the first file created.

```
$ cat file1.txt file2.txt
```

Output: This is a sample text file.
This is the first file created.
This is a sample text file.
This is the second file created.

The command can also be used to create a file as described below. The meaning and significance of the ">" symbol will be discussed later.

```
$ cat > newfile.txt
```

Output: This is a new file being created.

Use the redirections to save contents into a file.

To indicate end of input, press <Ctrl-d>

Instructor Notes:

cat Command

The cat command can be used to display contents of more than one file.

- It displays contents of chap2 immediately after displaying chap1.

```
$ cat chap1 chap2
```

Instructor Notes:

Input and Output Redirection

Standard Input : Keyboard

Standard Output: Monitor

Standard Error : Monitor

Redirection operators:

- < : Input Redirection
- > : Output Redirection
- 2> : Error Redirection
- >> : Append Redirection

Input and Output Redirection:

Many commands work with **character streams**. The default is the keyboard for input (standard input, file number 0), and terminal for the output (standard output, file number 1). In case of any errors, the system messages get written to standard error (file number 2), which defaults to a terminal.

UNIX treats each of these streams as files, and these files are available to every command executed by the shell. It is the shell's responsibility to assign sources and destinations for a command. The shell can also replace any of the standard files by a physical file, which it does with the help of **metacharacters** for redirection.

Instructor Notes:

Redirection

Input redirection: Instead of accepting i/p from standard i/p(keyboard) we can change it to file.

- **Example:** $cat < myfile will work same as $cat myfile
- < indicates, take i/p form myfile and display o/p on standard o/p device.

Output redirection: To redirect o/p to some file use >

- **Example:** $cat < myfile > newfile
- The above command will take i/p from myfile and redirect o/p to new file instead of standard o/p (monitor).

Redirection:

In order to take the input from a file (instead of standard i/p), the character **<** is used.

Example: $ cat < file1.txt

This is example of i/p redirection.

To redirect the output to a file, **>** is used. If outfile does not exist, it is first created; otherwise, the contents are overwritten. In order to append output to the existing contents, **>>** is used.

In the following command, the **cat** command will take i/p from file **file1.txt** and send the o/p to **result file**. If result file exists, then contents will be overwritten, otherwise new result file will be created.

Example:

$ cat < file1.txt > result is same as $cat file1.txt > result

$ cat result

2 12 60

Instructor Notes:

Redirection

- $ cat < file1.txt > result is same as $cat file1.txt > result.

```
$ cat result
```

Output: 2 12 60

- >> is append redirection
- The given command will append the contents of file1.lst in result file.

```
$ cat < file1.lst >> result
$ cat result
```

Output: 2 12 60
4 4 8

Note:
>> - is append redirection
If you want to retain previous contents of a file and append new contents to a file then use append redirection operator.

```
$ cat < file1.lst >> result
$ cat result
  2  12  60
  4  4  8
```

Instructor Notes:

cat file exist/not exist

Consider an example of cat –(file exist/not exist):

```
$ cat abc.txt > pqr.txt 2> errfile.txt
```

- If file abc.txt exists:
  - Then contents of the file will be sent to pqr.txt. Since no error has occurred nothing will be transferred to errfile.txt.
- If abc.txt file does not exist:
  - Then the error message will be transferred to errfile.txt and pqr.txt will remain empty.

cat file ext/not exit:
As shown in the above slide, it is possible to combine redirection operators on a single command line. The order of the redirection symbols does not affect the working of the command.

To redirect the standard error, 2> is used.

```
$ cat xyzfile
```

```
$ cat xyzfile 2> errfile
$ cat errfile
```

cat: cannot open xyzfile: No such file or directory (error 2)

```
$ cat xxfile 2>> errfile
$ cat errfile
```

cat: cannot open xyzfile: No such file or directory (error 2)

cat: cannot open xyzfile: No such file or directory (error 2)
cat: cannot open xxfile: No such file or directory (error 2)

Instructor Notes:

cp Command (copy file)

The cp (copy file) command copies a file or group of files.

The following example copies file chap1 as chap2 in test directory.

- Example:

```
$ cp chap1 temp/chap2
Option - i (interactive)
        $cp  - i  chap1    chap2
        cp: overwrite chap2   ? y
Option -r (recursive) to copy entire directory
$cp  -  r   temp   newtemp
```

cp Command (Copying Files):

The copy command copies a file or groups of files.

```
$ cp newfile.txt anotherfile.txt
$ cp newfile.txt testdir1/nfile1.txt
```

Copy all files with extension txt:

```
$ cp *.txt *.dat
```

All files with extension will be copied with same name but extension will change to dat.

Example:

abc.txt will be copied as abc.dat

pqr.txt will be copied as pqr.dat

Files cannot be copied if they are read protected, or if destination file/directory is write protected.

Copying file using redirection:

```
$cat newfile.txt > anotherfile.txt
```

The result of above command is same as $ cp newfile.txt anotherfile.txt

Instructor Notes:

rm Command (delete file)

The rm (remove file) command is used to delete files:

```
$ rm chap1       chap2       chap3
      $ rm  *
        Are you sure? y
Option - i (interactive delete)
      $ rm  -  i     chap1       chap2
      chap1 : ? Y
      chap2  :?  Y
Option   -   r (recursive delete)    (Avoid using this option)
```

rm Command (Removing Files) :

This command is used to delete files. There are options for **interactive (-i) delete** and **recursive (-r) delete.** The **–r** option will delete files from subfolders also.

```
$ rm newfile.txt
Option   -   r (recursive delete)
$ rm  -r *
(Warning: Pl. do not use this option)
```

Instructor Notes:

mv Command

The mv command is used to rename file or group of files as well as directories.

```
$ mv  chap1  man1
```

The destination file, if existing, gets overwritten:

- Example: $ mv temp doc
- Example: $ mv chap1 chap2 chap3 man1
  - It will move chap1, chap2 & chap3 to man1 directory

mv Command (Renaming Files):

Files as well as directories (belonging to the same parent) can be renamed using this command.

```
$ mv anotherfile.txt newfile.txt
$ mv testdir1 testdir2
$ ls -l
```

```
total 12
-rw-r--r--  3 deshpavn group      60 Mar 29 10:43 file1.txt
-rw-r--r--  1 deshpavn group      61 Mar 29 10:44 file2.txt
-rw-r--r--  1 deshpavn group      60 Mar 29 10:44 file3.txt
drwx------  2 deshpavn group     512 Mar 14 10:00 mail
-rw-r--r--  1 deshpavn group     126 Mar 29 10:54 newfile.txt
drwxr-xr-x  3 deshpavn group     512 Mar 29 10:58 testdir2
```

Instructor Notes:

wc Command

The wc command counts lines, words, and character depending on option.

It takes one or more filename as arguments.

no filename is given or - will accept data from standard i/p.

```
$ wc infile
3  20  103 infile
$wc      or   $wc -
This is standard input
press ctrl-d to stop
```

- **Output:** 2 8 44

Line, word, and character counting – wc

This command can be used to count the number of lines (-l option),words (-w option), or characters (-c option) for one or more files. If we specify multiple files, then the list of files should be separated by space. If no file name is specified, then it will accept data from standard i/p, that is from the keyboard.

Example:

```
$ wc file1.txt
```

2 12 60 file1.txt

```
$ wc -lw file1.txt
```

212 file1.txt

Instructor Notes:

wc Command

```
$ wc  infile test
```

Output: 3 20 103 infile
10 100 180 test
13 120 283 total

```
$ wc  - l infile
```

Output: 3 infile

```
$ wc  - wl  infile
```

Output: 20 3 infile

The following command will take i/p from infile and send o/p to result file

```
$ wc < infile > result
$ cat result
```

Output: 2 12 60

wc Command with Redirection:

In order to take the input from a file (instead of standard i/p), the character < is used.

```
$ wc < file1.txt
```

Output: 2 12 60

To redirect the output to a file, > is used. If outfile does not exist, it is first created; otherwise, the contents are overwritten. In order to append output to the existing contents, >> is used.

In following command, wc will take i/p from file file1.txt and send the o/p to result file. If result file exists, contents will be overwritten, otherwise new result file will be created.

```
$ wc < file1.txt > result
$ cat result
```

Output: 2 12 60

```
$ wc < cfile1.lst >> result
$ cat result
```

Output: 2 12 60
4 4 8

Instructor Notes:

cmp Command

cmp Command:

```
$ cmp file1.txt file2.txt
file1.txt file2.txt differ: char 41, line 2
$ cmp file1.txt file1.txt
```

cmp Command (Comparing Files):

Using the cmp Command:

The cmp command can be used to compare if two files are matching or not. The comparison is done on a byte by byte basis. In case files are identical, no message is displayed. Otherwise, locations of mismatches are echoed.

```
$ cmp file1.txt file2.txt
        file1.txt file2.txt differ: char 41, line 2
  $ cmp file1.txt file1.txt
```

Instructor Notes:

comm Command

comm Command:

- The comm command compares two sorted files. It gives a 3 columnar output:
  - First column contains lines unique to the first file.
  - Second column contains lines unique to the second file.
  - Third column displays the common lines.

comm Commands (Comparing Files):

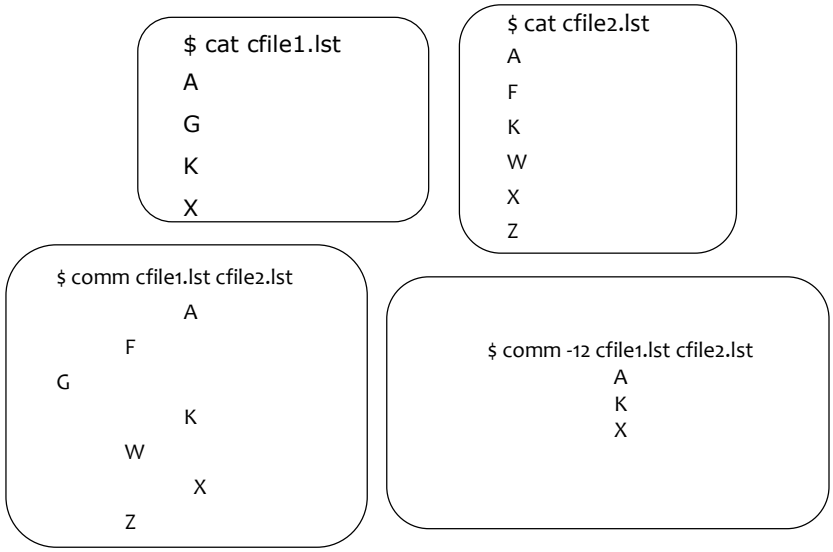
The comm command compares two sorted files. It gives a 3 columnar output.
First column contains lines unique to the first file.

Second column contains lines unique to the second file.
Third column displays the common lines.

Selective column output can be obtained by using options –1, -2 or –3. It would drop the column(s) specified from the output.

Instructor Notes:

comm Command



In above example
$ comm cfile1.lst cfile2.lst
This command is showing output in 3 columns
First column display lines unique to the cfile1.lst.
Second column display lines unique to the cfile2.lst.
Third column displays the lines common in both files.

$ comm -12 cfile1.lst cfile2.lst
This command will hide first and second column and display only 3 rd column i.e lines common in both files.

Instructor Notes:

diff Command

The diff command is used to display the file differences. It tells the lines of one file that need to be changed to make the two files identical.

- Example:

```
$ diff cfile1.lst cfile2.lst
2c2
< G
> F
3a4
> W
4a6
> Z
```

diff Command:

The file differences can be displayed using the diff command. It tells which lines of one file need to be changed to make the two files identical.

Example: Using the diff command

```
$ diff cfile1.lst cfile2.lst
2c2         change line 2 of first file which is line 2 in 2nd file
< G         replacing this line
---         with
> F         this line
3a4         append after line 3 in first file
> W         this line
4a6         append after line 4 in first file
> Z         this line
```

Instructor Notes:

tr Command

The tr command accepts i/p from standard input.

This command takes two arguments which specify two character sets.

The first character set is replaced by the equivalent member in the second character set.

The –s option is used to squeeze several occurrences of a character to one character.

tr Command:
It accepts i/p from standard i/p.
Hence to give input to tr command we have to use i/p redirection operator.
Example:

```
$tr –s” “ file1.txt
```

This is wrong command it will not take i/p from file1.txt.
The correct way of giving i/p is as follows:

```
$tr –s“  “  < file1.txt
```

Instructor Notes:

tr Command

Example 1: To squeeze number of spaces by single space:

```
$ tr –s “ ”  < file1.txt
```

Example 2: To convert small case into capital case:

```
$ tr “[a-z]” “[A-Z]” < file1.txt
ONE
TWO
THREE
FOUR
```

tr Command:

Example: To convert small case into capital case:

```
$ tr “[a-z]” “[A-Z]” < file1.txt
```

The tr command will replace a with A , b with B, and so on.

Instructor Notes:

more Command

The more command, from the University of California, Berkeley, is a paging tool.

The more command is used to view one page at a time. It is particularly useful for viewing large files.

Syntax for more command is as follows:

```
more <options> <+linenumber> <+/pattern> <filename(s)>
```

Example: To display file1.txt one screenful at a time

```
$ more file1.txt
```

more Command (Viewing a file one screen at a time):

This command from the University of California, Berkeley, is a paging tool – it can be used to view one page at a time. It is particularly useful for viewing large files.

Syntax of this is as follows:

```
more <options> <+linenumber> <+/pattern> <filename(s)>
```

There are a number of options available with the more command. Some of them are listed below:

Command	Description
Spacebar	Displays next screen
Ks	Skips K lines forward
Kf	Skips K screens forward
=	Displays current line number
:f	Displays current file name and line number
K:n	Skips to Kth next file specified on command line
K:p	Skips to Kth previous file specified on command line
/pattern	Searches forward for specified pattern
.	Repeats previous command
Q	Exits command

Instructor Notes:

chmod Command (Alter File Permissions)

The chmod command is used to alter file permissions:

Syntax:

```
chmod <category> <operation> <permission> <filenames>
```

Category	Operations	Attribute
u-user	+assigns permission	r-read
g-group	-remove permission	w-write
o-others	=assigns absolute permission	x-execute
a-all		

In UNIX operating system every file is owned by the user who created that file. Only owner or Superuser can change the file permissions using chmod command.

Superuser

In UNIX the superuser is responsible for system administration. **root** is the conventional name of the superuser who has all rights or permissions.

It is never good practice for anyone to use root as their normal user account, since simple typographical error in entering commands can cause major damage to the system. It is advisable to create a normal user account instead and then use the su command to switch to root user when necessary.

Instructor Notes:

chmod Command (Alter File Permissions)

Example 1:

```
$ chmod u+x note
$ ls - l note
      -rwx r-- r --1 ……    note
```

Example 2:

```
$ chmod ugo+x          note
$ ls - l note
  -rwxr-xr-x ……       note
```

- When we use + symbol, the previous permissions will be retained and new permissions will be added.
- When we use = symbol, previous permissions will be overwritten.

chmod Command (Working with file permissions):

The **chmod** command is used to change the file permissions. Permissions can be specified in two ways:
by using symbolic notation, or
by using octal numbers

The table given below describes the category, operation, and attributes required by the **chmod** command:

Category	Operation	Attribute & value
u-user	+ assign permission	r-read (4)
g-group	- remove permission	w-write (2)
o-others	= assign absolute permission	x-execute (1)
A-all		

In **octal notation**, r is 4, w is 2, and x is 1. To give read and write permission to only user, we use number 600. 1st number indicates permissions for user(4+2), 2nd number for group, and 3rd number as others.

The following command will give read, write, and execute permissions to user(4+2+1=7), and read, write permissions to group(4+2), and execute (1) permission to others.

Let us consider some examples.

Instructor Notes:

chmod Command (Alter File Permissions)

Example 3:

```
$ chmod u-x, go+r      note
      $ chmod u+x      note      note1      note2
      $ chmod o+wx     note
      $ chmod ugo=r    note
```

chmod Command (Working with file permissions):

Example 1:

```
$ chmod 761 file1.txt
$ ls -l file1.txt
-rw-rw-r--  3 deshpavn group       60 Mar 29 10:43 file1.txt
```

Example 2:

```
$ chmod ugo-w file1.txt
$ ls -l file1.txt
-r--r--r--  3 deshpavn group       60 Mar 29 10:43 file1.txt
```

Example 3:

```
$ chmod a+w file1.txt
$ ls -l file1.txt
-rw-rw-rw-  3 deshpavn group       60 Mar 29 10:43 file1.txt
```

Example 4:

```
$ chmod o-w,ug+x file1.txt file2.txt
$ ls -l file1.txt file2.txt
-rwxrwxr--  3 deshpavn group       60 Mar 29 10:43 file1.txt
-rwxr-xr--  1 deshpavn group       61 Mar 29 10:44 file2.txt
```

Example 5:

```
$ chmod 644 file1.txt file2.txt
$ ls -l file1.txt file2.txt
-rw-r--r--  3 deshpavn group       60 Mar 29 10:43 file1.txt
-rw-r--r--  1 deshpavn group       61 Mar 29 10:44 file2.txt
```

Instructor Notes:

chmod Command (Alter File Permissions)

Octal notation:

- It describes both category and permission.
- It is similar to = operator (absolute assignment).
  - read permission: assigned value is 4
  - write permission: assigned value is 2
  - execute permission: assigned value is 1
- Example 1:

```
$ chmod  666 note
```

- It will assign read and write permission to all.

chmod Command (Working with file permissions):

Setting Permissions:

The chmod command uses a string, which describes the permissions for a file, as an argument.

The permission description can be in the form of a number that is exactly three digits. Each digit of this number is a code for the permissions level of three types of people that might access this file:

Owner
Group (a group of users where owner is part of)
Others (anyone else browsing around on the file system)

The value of each digit is set according to the rights that each of the types of people listed above have to manipulate that file.

Permissions are set according to numbers**. Read is 4. Write is 2. Execute is 1**. The sums of these numbers, give combinations of these permissions:

0 = no permissions whatsoever; this person cannot read, write, or execute the file
1 = execute only
2 = write only
3 = write and execute (1+2)
4 = read only
5 = read and execute (4+1)
6 = read and write (4+2)
7 = read and write and execute (4+2+1)

Instructor Notes:

chmod Command (Alter File Permissions)

- Example 2:

```
$ chmod  777 note
```

  - It will assign all permissions to all.

- Example 3:

```
$ chmod  753 note
```

chmod Command (Working with file permissions):

Permissions are given using these digits in a sequence of three: one for owner, one for group, one for world.

Let us look at how I can make it impossible for anyone else to do anything with my apple.txt file but me:

```
$ chmod 700 apple.txt
```

If someone else tries to look into apple.txt, they get an error message:

```
$ cat apple.txt
cat: apple.txt: Permission denied
$
```

If I want other people to be able to read apple.txt, I would set the file permissions as shown below:

```
$ chmod 744 apple.txt
$
```

Detecting File Permissions:

You can use the ls command with the -l option to show the file permissions set. For example, for apple.txt, I can do a change as follows:

```
$ ls -l apple.txt
-rwxr--r--   1 december december       81 Feb 12 12:45 apple.txt
```

The sequence -rwxr--r-- tells the permissions set for the file apple.txt. The first - tells that apple.txt is a file. The next three letters, rwx, show that the owner has read, write, and execute permissions. Then the next three symbols, r--, show that the group permissions are read only. The final three symbols, r--, show that the others permissions are read only.

Instructor Notes:

mkdir Command

The mkdir command creates a directory.

- Example: 1:

```
$ mkdir doc
```

- Example 2:

```
$ mkdir doc    doc/example        doc/data
```

- Example 3:

```
$ mkdir        doc/example        doc
```

  - It will give error - Order important.

mkdir Command (Creating directory):

A single directory, or a number of subdirectories, can be created using the mkdir command. A directory will not get created if there is already another directory by the same name under the parent directory.

Besides, appropriate permissions will be required.

Example:

```
$ mkdir newdir1
$ ls -F
file1.txt
file2.txt
file3.txt
mail/
newdir1/
newfile.txt
testdir2/
$ mkdir newdir2 newdir2/subdir1 newdir2/subdir2
$ ls -l newdir2
total 4
drwxr-xr-x   2 deshpavn group       512 Mar 29 11:09 subdir1
drwxr-xr-x   2 deshpavn group       512 Mar 29 11:09 subdir2
```

Instructor Notes:

rmdir Command

The rmdir command is used to remove directory.

Only empty dir can be deleted.

More than one dir can be deleted in a single command.

Command should be executed from at least one level above in the hierarchy.

rmdir Command (Removing directory):

The **rmdir** command can be used to delete one or more directories.

Any directory can be deleted only if it is empty. It is important to note that the command needs to be issued from a directory, which is hierarchically above the directory to be deleted.

Example:

```
$ rmdir newdir1
```

Instructor Notes:

rmdir Command

Example 1:

```
$ rmdir doc
```

Example 2:

```
$ rmdir doc/example doc
```

Example 3:

```
$ rmdir doc  doc/example
```

- It will give error.

Instructor Notes:

Internal and External Commands:

External commands
- A new process will be set up
- The file for external command should be available in BIN directory
- E.g – cat, ls , Shell scripts

Internal commands
- shell's own built in statements, and commands
- No process is set up for such commands.
- E.g cd , echo

Internal and External Commands:

The shell recognizes two types of commands – external and internal.

```
$ rmdir newdir1
```

External commands

- These are commands like cat, ls, and so on or utilities. Shell scripts also come under the category of external commands.
- A new process will be set up for the external commands.
- The file for external command should be available in BIN directory

Internal commands

- These are the shell's own built in statements, and commands such as cd, echo, and so on.
- No process is set up for such commands. For external command it is necessary that some commands are built into the shell itself and no process is set up. That is because it is very difficult or sometimes impossible to implement some commands as external commands.

Instructor Notes:

Summary

In this lesson, you have learnt:

- UNIX organizes files in hierarchical manner.
- File access can be secured using different file permissions.
- < - Input Redirection
- > - Output Redirection
- 2> - Error Redirection
- chmod command is used to change file permissions.

Instructor Notes:

Review Questions

Question 1: To copy all files with extension txt to mydir directory ___ command is used, if mydir is parent directory of current directory.

- Option 1: cp *.txt ..
- Option 2: cp *.txt ../mydir
- Option 3: cp mydir *.txt

Question 2: 2> symbol is used as error redirection

- True / False

Question 3: cd . changes the directory to ___.

Question 4: Which of the following command will give only read permission to all for file file1.txt?

- Option 1: chmod a=r file1.txt
- Option 2: chmod a+r file1.txt
- Option 3: Chmod 666 file1.txt

Instructor Notes:

Review – Match the Following

1. To change directory to home directory	a. rm *.dat
2. To remove all files with extension *.dat	b. cat <abc.txt
3. To display contents of file abc.txt	c. cat > abc.txt
4. To create abc.txt file	d. cd
	e. cd \
	f. mkdir mydir