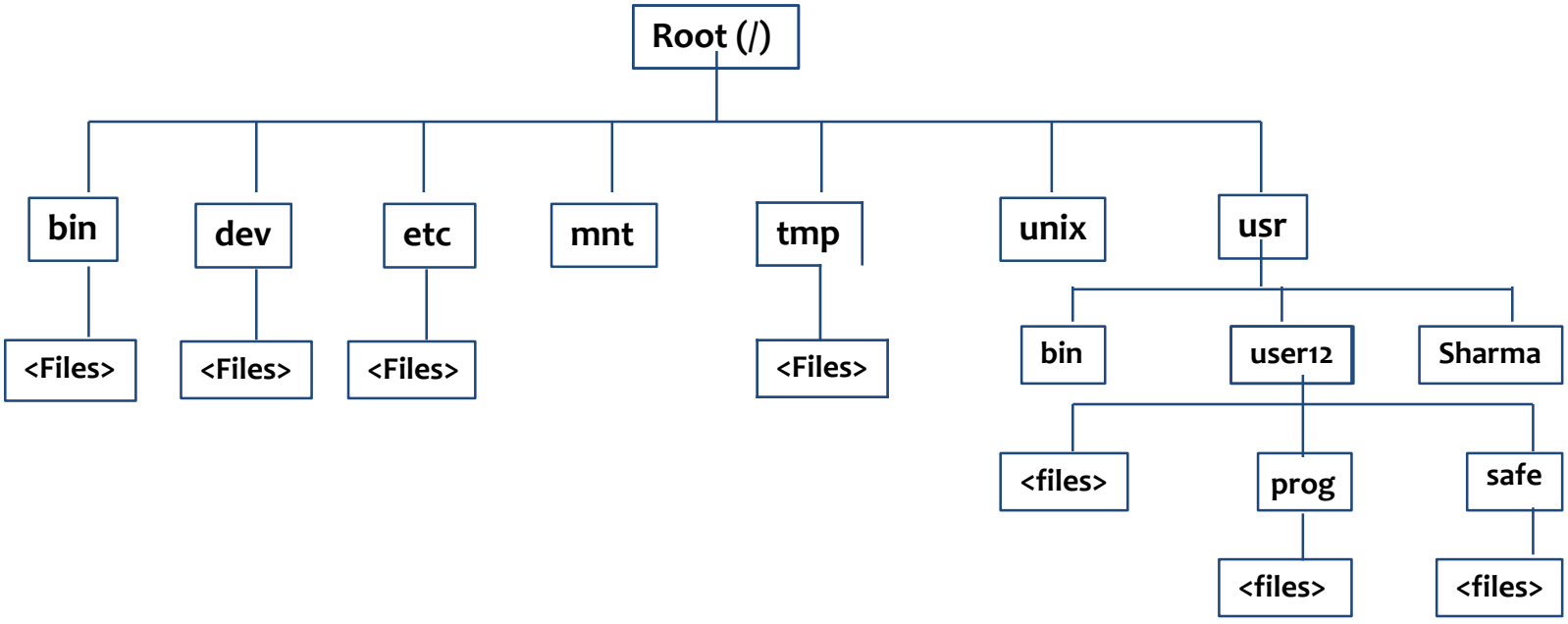
UNIX

UNIX File System

File System Structure



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

File System

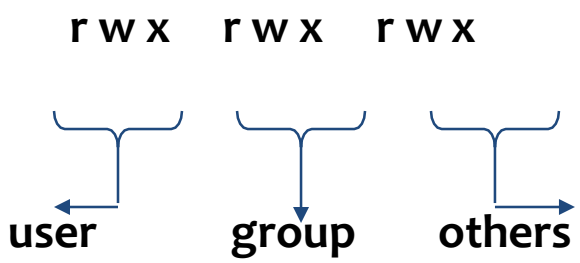
- /tmp **: used by some UNIX utilities especially vi and by user to store temporary files**
- /usr **: contains all the files created by user, including login directory**
- /unix **: kernel**
- **Release V:**
  - It does not contain / bin.
  - It contains / home instead of /usr.

File Types in UNIX

- **We have the following file types in UNIX:**
  - Regular File
  - Directory File
  - Device File
  - Pipe file
  - Link file

File Permissions in UNIX

- **File Access Permissions**



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)

pwd Command

- **The** pwd **command checks current directory.**

```
$ pwd
```

  - **Output:** /usr/user12

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/user12

```
$ cd Prog
$ pwd
```

- **Output:** /usr/user12/Prog

cd Command

- **Moving one level up:**

```
$ cd ..
```

Switching to home directory:

```
$ cd
```

- **Switching to /usr/sharma:**

```
$ cd  /usr/Sharma
```

- **Switching to root directory:**

```
$ cd /
```

logname Command

- **The** logname **command checks the login directory.**

```
$ logname
```

Output: user12

ls Command

- **The ls command lists the directory contents.**
- **Example:**

```
$ ls
```

Output:

a.out
pack1
pack2
test
test.c

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

- **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

- **Example:**

```
$ ls – l
```

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

```
-rw-rw-rw-    1 user12 group 44 May 9 09:08 dept.h
-rw-rw-rw-    1 user12 group 212 May 9 09:08 dept.q
-rw-rw-rw-    1 user12 group 154 May 9 09:08 emp.h
```

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

- **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

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**

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

- **The** cat **command can be used to display contents of more than one file.**
  - It displays contents of pack2 immediately after displaying pack1.

```
$ cat pack1 pack2
```

Input and Output Redirection

- **Standard Input : Keyboard**
- **Standard Output : Monitor**
- **Standard Error : Monitor**

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

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

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

```
$ cat result
```

Output: 2 15 20

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

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

Output: 2 15 20
4 4 8

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.

tac

tac is practically the reverse version of cat command (also spelled backwards) which prints each line of a file starting from the bottom line and finishing on the top line to your machine standard output.

$tac myfile.txt
This is last line four
This is line three
This is line two
This is first line one

Usually use of tac command is, that it can provide a great help in order to debug log files,
reversing the chronological order of log contents.
$ tac /var/log/auth.log

Or to display the last lines

$ tail /var/log/auth.log | tac

cp Command (copy file)

- **The** cp **(copy file) command copies a file or group of files.**
- **The following example copies file pack1 as pack2 in test directory.**
  - Example:

```
$ cp pack1 temp/pack2
Option - i (interactive)
      $cp  - i  pack1    pack2
      cp: overwrite pack2          ? y
Option -r (recursive) to copy entire directory
$cp  -  r   temp   mytemp
```

rm Command (delete file)

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

```
$ rm pack1    pack2     pack3
      $ rm   *
        Are you sure? y
Option - i (interactive delete)
      $ rm - i pack1        pack2
      pack1 : ? Y
      pack2 :?  Y
Option   -   r (recursive delete)     (Avoid using this option)
```

mv Command

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

```
$ mv   pack1 man1
```

- **The destination file, if existing, gets overwritten:**
  - Example: $ mv temp doc
  - Example: $ mv pack1 pack2 pack3 man1
    - It will move pack1, pack2 & pack3 to man1 directory

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

- **Output:** 2 8 44

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

cmp Command

- **cmp Command:**

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

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 Command

```
$ cat cfile1.lst
A
k
p
X
```

```
$ cat cfile2.lst
A
F
K
W
X
Z
```

```
$ comm cfile1.lst cfile2.lst
                A
        F
p
                K
        W
                 X
        Z
```

```
$ comm -12 cfile1.lst cfile2.lst
A
K
X
```

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
< p
> F
3a4
> W
4a6
> Z
```

Uname

Is used to determine details like kernel name, version, hostname, etc of the Linux box you are using.

Even though you can find all these details in respective files present under the proc filesystem, it is easier to use uname utility to get these information quickly.

The basic syntax of the uname command is :

uname [OPTION]...

Option

Uname –r

- **To see the version of kernel.**

uname without any option
When the ‘uname’ command is run without any option then it prints just the kernel name. So the output below shows that its the ‘Linux’ kernel that is used by this system.

$ uname
Linux

To displays the kernel name.

$ uname -s
Linux

Get the network node host name using -n option

$ uname -n
dev-server

Get kernel release using -r option
to fetch the kernel release information(version information).

$ uname -r
2.6.32-100.28.5.el6.x86_64

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

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

```
$ tr –s “ ” < file1.txt
```

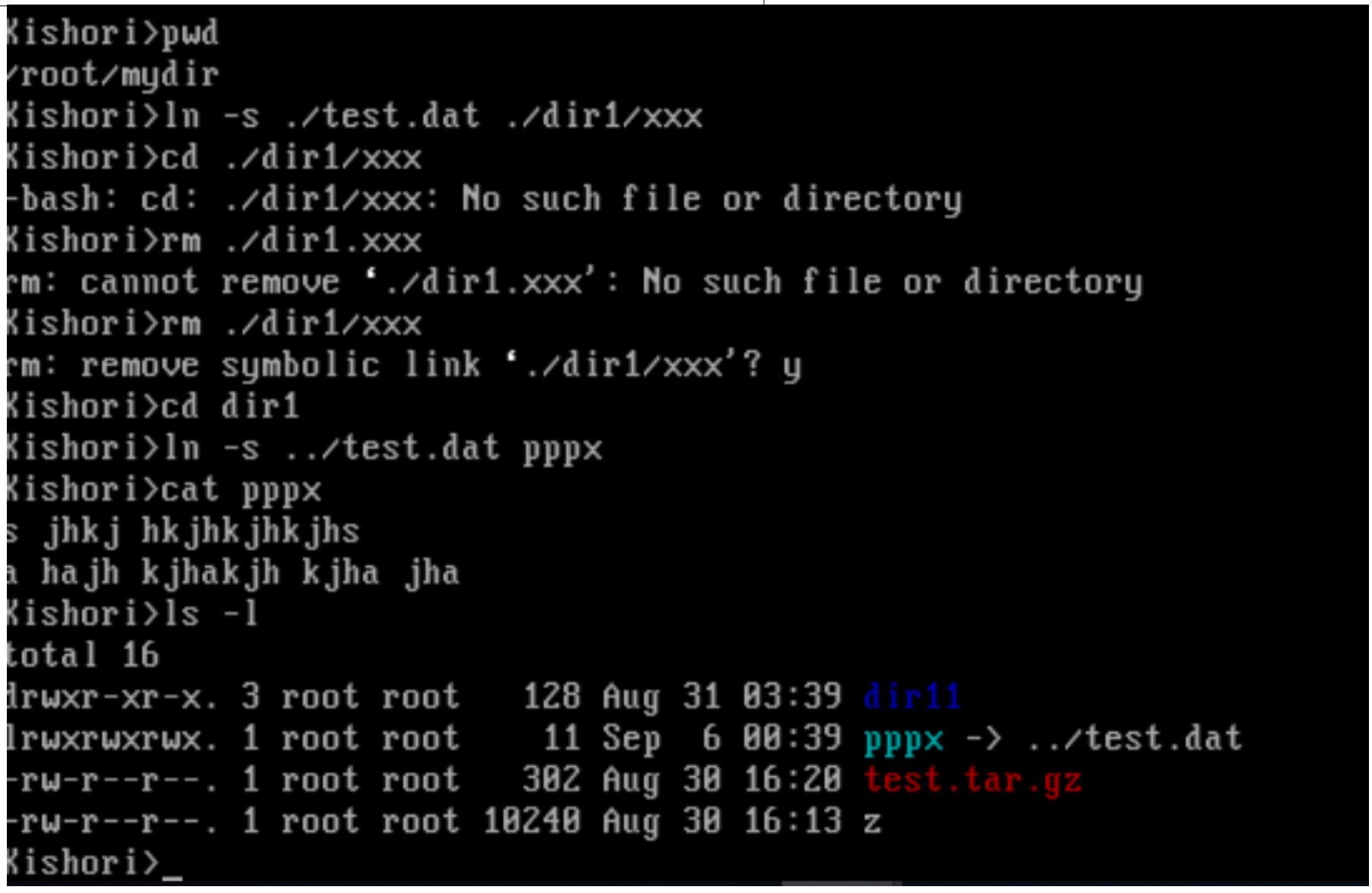
- Example 2: **To convert small case into capital case:**

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

$ ln –s ../test.dat ./dir11/ppp

For symbolic link if you try to access original file by using symbolic link name then it gives error is you have not specified proper relative path

If you do not specify relative path then you cannot read original file using symbolic link because it tries to find original file using relative path


Kishori>pwd
/root/mydir
Kishori>ln -s ./test.dat ./dir1/xxx
Kishori>cd ./dir1/xxx
-bash: cd: ./dir1/xxx: No such file or directory
Kishori>rm ./dir1.xxx
rm: cannot remove ‘./dir1.xxx’: No such file or directory
Kishori>rm ./dir1/xxx
rm: remove symbolic link ‘./dir1/xxx’? y
Kishori>cd dir1
Kishori>ln -s ../test.dat pppx
Kishori>cat pppx
s jhkj hkjhkjhkjhs
a hajh kjhakjh kjha jha
Kishori>ls -l
total 16
drwxr-xr-x. 3 root root 128 Aug 31 03:39 dir11
lrwxrwxrwx. 1 root root 11 Sep 6 00:39 pppx -> ../test.dat
-rw-r--r--. 1 root root 302 Aug 30 16:20 test.tar.gz
-rw-r--r--. 1 root root 10240 Aug 30 16:13 z
Kishori>_

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
```

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		

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 (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 (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 (Alter File Permissions)

- Example 2:

```
$ chmod  777  note
```

  - It will assign all permissions to all.

- Example 3:

```
$ chmod  753  note
```

mkdir Command

- **The** mkdir **command creates a directory.**

```
$ mkdir mytemp
```

```
$ mkdir dir1     dir1/example          dir1/data
```

```
$ mkdir          dir1/example          dir1
```

  - It will give error - Order important.

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

```
$ rmdir doc
```

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

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

- It will give error.

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

Whatis

- **whatis command**
  - **displays short manual page descriptions.**

Each manual page has a short description available within it.

Whatis searches the manual page names and displays the manual page descriptions of any name matched.

name may contain wildcards (-w) or be a regular expression (-r).

Index databases are used during the search and are updated by the mandb program.

whereis

Whereis

The whereis command lets users locate binary, source, and manual page files for a command. Following is its syntax:

whereis [options] name...

Suppose you want to find the location for, the whereis command itself.

whereis whereis

The whereis command also produces paths for bonary files, manual pages and source code

How to specifically search for binaries, manuals, or source code?

If you want to search specifically for, binary, then you can use the –b option. For example:
whereis -b cp

Similarly, the -m and -s options are used in case you want to find manuals and sources.

How to limit whereis search as per requirement?

By default whereis tries to find files from hard-coded paths, which are defined with glob patterns.

However, if you want, you can limit the search using specific command line options.

if you want whereis to only search for binary files in /usr/bin, use the –B option.
whereis -B /usr/bin/ -f cp

Note: Since you can pass multiple paths this way, the -f command line option terminates the directory list and signals the start of file names.

Similarly, if you want to limit manual or source searches, you can use the -M and -S command line options.

Locate

The locate command is very easy to use. All you have to do is to pass it the filename you want to search.

locate [filename]

For example, if want to search for all filenames that have the string 'dir2' in them, then

```
$locate *dir2*
```

Note: The command 'locate dir2' (no asterisks) will also do as locate implicitly replaces the name you pass (say NAME) with *NAME*

How locate command works, or, why is it so fast

locate is so fast is because it doesn't read the file system for the searched file or directory name.

It actually refers to a database (prepared by the command updatedb) to find what user is looking.

While this is a good approach, it has its share of drawbacks.

The main issue is that after every new file or directory is created on the system, you need to update the tool's database for it to work correctly. Otherwise, the command will not be able to find files/directories that are created after the last database update.

For example, if I try finding files with names containing ‘myfile' string in the 'Downloads' directory of my system, the find command produces one result in the output:

But when I try performing the same search using the locate command, it produces no output.

This means that the database locate searches in wasn't updated after the file was created on the system. So, let's update the database, which can be done using the updatedb command

To do that:

sudo updatedb

To update database used in locate

sudo updated

And now when we run the same locate command again, it shows files in the output:

Similarly, after a file or directory has been removed, you need to make sure that the locate database has been updated, as otherwise, the command will keep showing the file in its output when searched.

locate command

if you want, you can make the tool suppress all this information, and just print the number or count of matching entries instead.

Use -c option.

Zcat

Zcat is a command line utility for viewing the contents of a compressed file without literally uncompressing it.

It expands a compressed file to standard output allowing you to have a look at its contents. In addition, zcat is identical to running gunzip -c command. In this guide, we will explain zcat command examples for beginners.

1. The first example shows how to view contents of a normal file using cat command, compress it using gzip command and view the contents of the zipped file using zcat as shown.

$ cat users.list

$ gzip users.list

$ zcat users.list.gz

UNIX

Filters

What is a Filter?

- **Filters are central tools of the UNIX tool kit.**
- **Commands work as follows:**
  - Accept some data as input.
  - Perform some manipulation on the inputted data.
  - Produce some output.
- **Most of them work on set of records, with each field of a record delimited by a suitable delimiter.**
- **When used in combination, they can perform complex tasks too.**

head Command

- **The** head **command, by default, will display the first 10 lines of a file.**
  - **Example 1:** To display fist 10 lines from file books:

```
$head books
```

  - **Example 2:** To display first 5 lines from file books:

```
$head –5 books
```

- **Single command can be used to display lines from more than one file.**

```
$ head -1 PuneEmp MumbaiEmp
```

tail Command

- **The** tail **command is useful to display last few lines or characters of the file.**
  - **Example 1:** To display last ten lines from books:

```
$tail books
```

  - **Example 2:** To display last seven lines:

```
$tail –7 books
```

  - **Example 3:** To display lines from the 10th line till end of the file:

```
$tail +10 books
```

  - **Example 4:** To display last 5 characters of the file:

```
$tail -5c books
```

cut Command

- **The** cut **command retrieves selected fields from a file.**

```
$ cut [options] <filename>
```

  - Options :
    - -c : selects columns specified by list
    - -f : selects fields specified by list
    - -d : field delimiter (default is tab)

cut Command

- **Example 1:** To display 2nd and 3rd field from file bookDetails.lst:

```
$ cut -d"|" -f2,3 bookDetails.lst
```

- **Example 2:** To display characters from 1st to 4th and 31st to 35th from file bookDetails.lst :

```
$ cut -c1-4,45-50 bookDetails.lst
```

paste Command

- **The paste command is used for horizontal merging of files.**

```
$paste <file1><file2><Enter>
```

  - Options : -d (Field delimiter)
  - **Example 1:** To paste enum.lst and ename.lst files:

```
$ paste enum.lst ename.lst
```

  - **Example 2:** To paste enum.lst and ename.lst files with '|' character as delimiter:

```
$ paste –d'I' enum.lst ename.lst
```

sort Command

- **The** sort **command is useful to sort file in ascending order.**

```
$sort <filename>
```

  - Options are:
    - -r : Reverse order
    - -n : Numeric sort
    - -f : Omit the difference between Upper and lower case alphabets
    - -t : Specify delimiter
    - -k : to specify fields as primary or secondary key
  - Example:

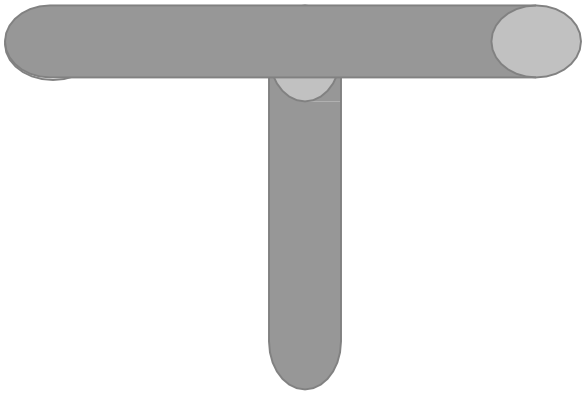
```
$ sort -t"|" +1 bookDetails.lst
$sort –k3,3 -k2,2 books
```

uniq Command

- **The** uniq **command fetches only one copy of redundant records and writes the same to standard output.**
  - –u option: It selects only non-repeated lines.
  - –d option: It selects only one copy of repeated line.
  - -c option: It gives a count of occurrences.
- **To find unique values, the file has to be sorted on that field.**
  - **Example:** To find unique values from file myfile.lst

```
$ uniq myfile.lst
```

tee Command



- To display contents of file books on screen as well as save it in the file:

```
$ tee user.txt < books
```

find Command

- **The** find **command locates files.**

```
find <path list> <selection criteria> <action>
```

  - To find the file named **.profile** starting at the root directory in the system **-print** specify the action:

```
$ find / -name .profile –print
```

  - To find the file named myfile starting at the root directory in the system

```
find / -type f -name "myfile" -print
```

grep Command

- **The syntax for** grep **command is as follows:**

```
grep <options> <pattern> <filename(s)>
```

  - The following example will search for the string Unix in the file **books.lst**. The lines which match the pattern will be displayed.

```
grep ‘Linux' books.lst
```

grep Command

- **Options of grep:**
  - c : It displays count of lines which match the pattern.
  - n : It displays lines with the number of the line in the text file which match the pattern.
  - v : It displays all lines which do not match pattern.
  - i : It ignores case while matching pattern.
  - -w : It forces grep to select only those lines containing matches that form whole words

grep Command

- To print all lines containing “mogara” regardless of case:

```
$grep -i mogra flower.txt
```

- To print all lines containing “mogara” as a word:

```
$grep -w mogara flower.txt
```

- To print all lines not containing “mogara”:

```
$grep -v mogara flower.txt
```

grep Command

Regular Expression:

Expression	Description
^ (Caret)	match expression at the start of a line, as in ^A.
$ (Question)	match expression at the end of a line, as in A$.
\ (Back Slash)	turn off the special meaning of the next character, as in \^.
[ ] (Brackets)	match any one of the enclosed characters, as in [aeiou]. Use Hyphen "-" for a range, as in [0-9].
[^]	match any one character except those enclosed in [ ], as in [^0-9].
. (Period)	match a single character of any value, except end of line.
* (Asterisk)	match zero or more of the preceding character or expression.
\{x,y\}	match x to y occurrences of the preceding.
\{x\}	match exactly x occurrences of the preceding.
\{x,\}	match x or more occurrences of the preceding.

grep Command

- **Examples of Regular Expression:**

Example	Description
grep "smile" files	**search *files* for lines with 'smile'**
grep '^smile' files	**'smile' at the start of a line**
grep 'smile$' files	**'smile' at the end of a line**
grep '^smile$' files	**lines containing only 'smile'**
grep '\^s' files	**lines starting with '^s', "\" escapes the ^**
grep '[Ss]mile' files	**search for 'Smile' or 'smile'**
grep 'B[oO][bB]' files	**search for BOB, Bob, BOb or BoB**
grep '^$' files	**search for blank lines**
grep '[0-9][0-9]' file	**search for pairs of numeric digits**

fgrep Command

- **The fgrep command is similar to grep command.**
- **Syntax:**

```
$fgrep [ -e pattern_list] [-f pattern-file] [pattern] [Search file]
```

- **The** fgrep **command is useful to search files for one or more patterns, which cannot be combined together.**
- **It does not use regular expressions. Instead, it does direct string comparison to find matching lines of text in the input.**

fgrep Command

- **Options of fgrep command:**
  - -e pattern_list :
    - It searches for a string in pattern-list.
  - -f pattern-file :
    - It takes the list of patterns from pattern-file.
  - pattern
    - It specifies a pattern to be used during the search for input.
    - It is same as grep command.
  - E.g To search books file for all patterns stored in mypattern file

    $ fgrep –f mypattern books.lst

egrep Command

- **The** egrep **command works in a similar way. However, it uses extended regular expression matching.**
  - Syntax:

```
egrep [ -e pattern_list ] [-f file ] [ strings ] [ file]
```

  - **Example:** To find all lines with name "aggrawal" even though it is spelled differently:

```
$ egrep '[aA]gg?[ar]+wal' stud.lst
```

Which command

- **The format of the which command is:**

  which command
- **which returns the location of binary, or executable, shell commands.**
- **The information provided by whichis useful for creating application launchers.**

  $which gedit
- **The above command returns the information /usr/bin/gedit.**

finger

- **finger looks up and displays information about system users.**
- **finger syntax**
- **finger [-lmsp] [user ...] [user@host ...]**
- **Options**
- **-s Displays the user's login name, real name, terminal name and write status (as a "*" after the terminal name if write permission is denied), idle time, login time, office location and office phone number.**

- **Login time is displayed as month, day, hours and minutes, unless more than six months ago, in which case the year is displayed rather than the hours and minutes.**

- **finger Examples**

  $finger -p user1
- **Display information about the user user1. Output will appear similar to the following:**
- **Login name: admin**
- **In real life: user one data**
- **On since Feb 11 23:37:16 on pts/7 from domain.uuu.com**
- **28 seconds Idle Time**
- **Unread mail since Mon Feb 12 00:22:52 2001**

- Gzip (GNU zip) is a compressing tool, which is used to truncate the file size. By default original file will be replaced by the compressed file ending with extension (.gz).
- To decompress a file you can use gunzip command and your original file will be back.
- Syntax:
- 1. gzip <file1> <file2> <file3>. . .
- 2. gunzip <file1> <file2> <file3>. . .
- 3.
- Example:
- 1. gzip file1.txt file2.txt
- 2. gunzip file1.txt file2.txt
- 3.
-
- Look at the above snapshot, the gzip command has compressed the files 'file1.txt' and 'file2.txt'. Compressed files are shown with the extension (.gz). While gunzip command has decompressed the same files and extension (.gz) is removed.

gzip options

Compressing Multi Files Together

If you want to compress more than one file together, you can use 'cat' and gzip command with pipe command.

Syntax:

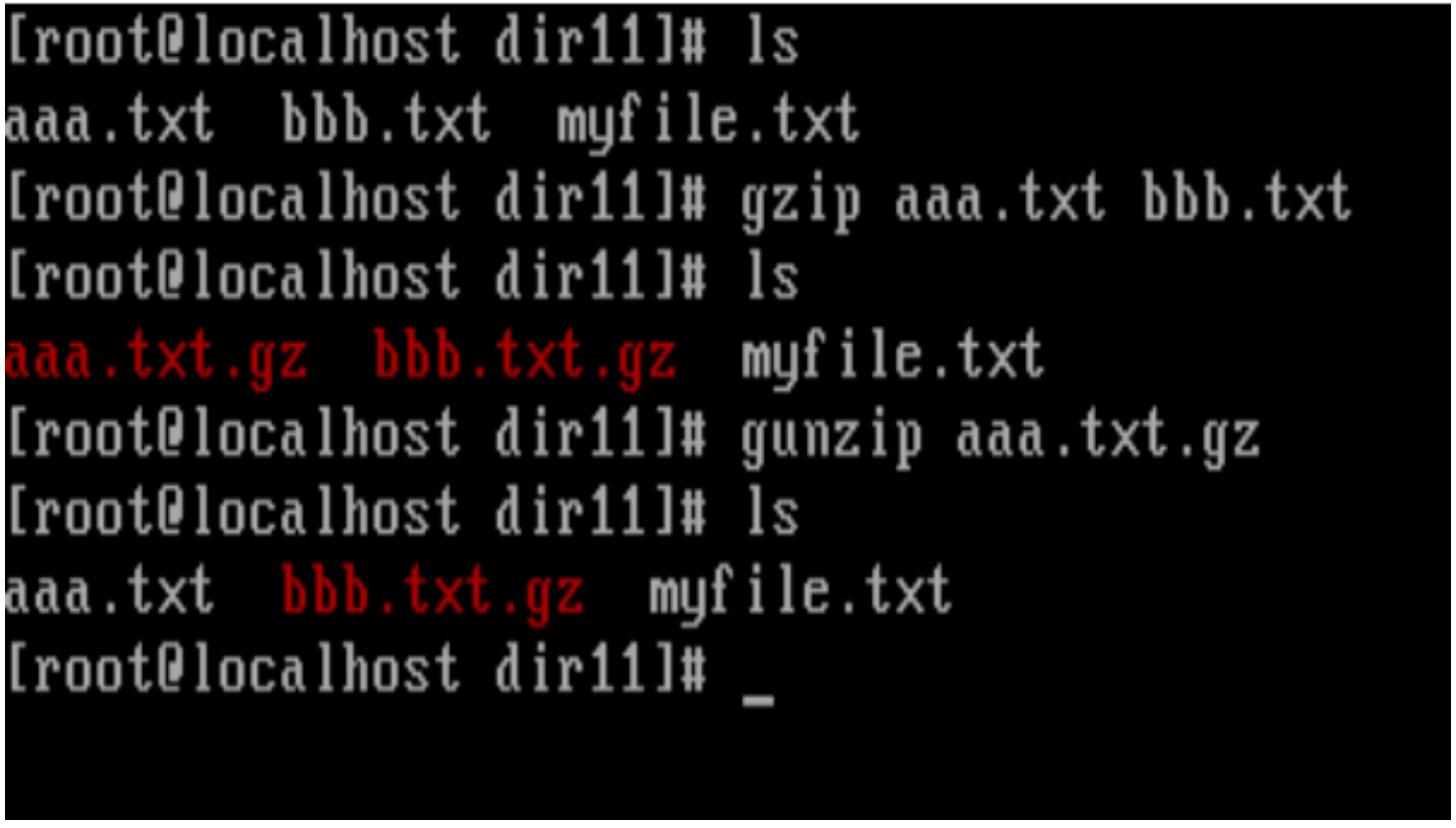
1. **cat <file1> <file2>. . | gzip > <newFile.gz>**

Example:

1. **cat file1.txt file2.txt | gzip > final.gz**

Look at the above snapshot, both the files ' file1.txt' and 'file2.txt' are compressed as 'final.gz'.

Each file will get replace by zip version of the file with extension .gz



How To Compress A Directory

The gzip command will not be able to compress a directory because it can only compress a single file. To compress a directory you have to use 'tar' command.
Hyphen (-) is not mandatory in 'tar' command.
'c' is to create,
'v' is for verbose, to display output,
'f' to mention destination of your output file,
'z' for specifying compress with gzip.

Syntax:

1. tar cf - <directory> | gzip > <directoryName>

OR

1. tar cvfz office.tar.gz office

Example:

1. tar cf - office | gzip > office.tar.gz

ZIP command in Linux with examples

ZIP is a compression and file packaging utility for Unix. Each file is stored in single .zip {.zip-filename} file with the extension .zip.

- **zip is used to compress the files to reduce file size and also used as file package utility. zip is available in many operating systems like unix, linux, windows etc.**
- **If you have a limited bandwidth between two servers and want to transfer the files faster, then zip the files and transfer.**
- **The zip program puts one or more compressed files into a single zip archive, along with information about the files (name, path, date, time of last modification, protection, and check information to verify file integrity). An entire directory structure can be packed into a zip archive with a single command.**
- **Compression ratios of 2:1 to 3:1 are common for text files. zip has one compression method (deflation) and can also store files without compression. zip automatically chooses the better of the two for each file to be compressed.**

The program is useful for packaging a set of files for distribution; for archiving files; and for saving disk space by temporarily compressing unused files or directories.

Syntax :
zip [options] zipfile files_list

Syntax for Creating a zip file:
$zip myfile.zip filename.txt

Extracting files from zip file
Unzip will list, test, or extract files from a ZIP archive, commonly found on Unix systems. The default behavior (with no options) is to extract into the current directory (and sub-directories below it) all files from the specified ZIP archive.

Syntax :

$unzip myfile.zip

Options :

1. -d Option: Removes the file from the zip archive. After creating a zip file, you can remove a file from the archive using the -d option.

Suppose we have following files in my current directory are listed below:

hello1.c

hello2.c

hello3.c

hello4.c

hello5.c

hello6.c

hello7.c

hello8.c

$zip myfile.zip *.c

$unzip myfile.zip

Syntax :
$zip –d filename.zip file.txt
Command :
$zip –d myfile.zip hello7.c
After removing hello7.c from myfile.zip file, the files can be restored with unzip command
Command:
$unzip myfile.zip
$ls command
Output :
hello1.c
hello2.c
hello3.c
hello4.c
hello5.c
hello6.c
hello8.c
The hello7.c file is removed from zip file

2.-u Option: Updates the file in the zip archive. This option can be used to update the specified list of files or add new files to the existing zip file. Update an existing entry in the zip archive only if it has been modified more recently than the version already in the zip archive.

Syntax:

$zip –u filename.zip file.txt

Suppose we have following files in my current directory are listed below:

hello1.c

hello2.c

hello3.c

hello4.c

Command :

$zip –u myfile.zip hello5.c

After updating hello5.c from myfile.zip file, the files can be restored with unzip command

Command:
$unzip myfile.zip
$ls command
Output :
hello1.c
hello2.c
hello3.c
hello4.c
hello5.c
The hello5.c file is updated to the zip file

touch

- **The touch command is the easiest way to create new, empty files.**
- **It is also used to change the timestamps (i.e., dates and times of the most recent access and modification) on existing files and directories.**
- **touch's syntax is**

  $touch [option] file_name(s)
- **When used without any options, touch creates new files for any file names that are provided as arguments (i.e., input data) if files with such names do not already exist.**
- **Touch can create any number of files simultaneously.**
- **Thus, for example, the following command would create three new, empty files named file1, file2 and file3:**

  $touch file1 file2 file3
- **A nice feature of touch is that, in contrast to some commands such as cp (which is used to copy files and directories) and mv (which is used to move or rename files and directories), it does not automatically overwrite (i.e., erase the contents of) existing files with the same name. Rather, it merely changes the last access times for such files to the current time.**

Several of touch's options are specifically designed to allow the user to change the timestamps for files. For example, the -a option changes only the access time, while the -m option changes only the modification time. The use of both of these options together changes both the access and modification times to the current time, for example:

touch -am file3

The -r (i.e., reference) option followed directly by a space and then by a file name tells touch to use that file's time stamps instead of current time. For example, the following would tell it to use the times of file4 for file5:

touch -r file4 file5

The -B option modifies the timestamps by going back the specified number of seconds, and the -F option modifies the time by going forward the specified number of seconds. For example, the following command would make file7 30 seconds older than file6.

touch -r file6 -B 30 file7

The -d and -t options allow the user to add a specific last access time. The former is followed by a string (i.e., sequence of characters) in the date, month, year, minute:second format, and the latter uses a [[CC]YY]MMDDhhmm[.ss] format. For example, to change the last access time of file8 to 10:22 a.m. May 1, 2005, 1 May 2005 10:22 would be enclosed in single quotes and used as follows, i.e.,:

touch -d '1 May 2005 10:22' file8

Partial date-time strings can be used. For example, only the date need be provided, as shown for file9 below (in which case the time is automatically set to 0:00):

touch -d '14 May' file9

- Just providing the time, as shown below, automatically changes the date to the current date:
- touch -d '14:24' file9
- The most commonly used way to view the last modification date for files is to use the ls command with its -l option. For example, in the case of a file named file10 this would be
- ls -l file10
- The complete timestamps for any file or directory can be viewed by using the stat command. For example, the following would show the timestamps for a file named file11:
- stat file11
- The --help option displays a basic list of options, and the --version option returns the version of the currently installed touch program.

Alias

Linux alias command: How to create and use Linux aliases

They let you define your own commands, or command shortcuts, so you can customize the command line, and make it work the way you want it to work.

To create search as alias name for grep
alias search=grep

Use search instead of grep at your Linux command line:
search 'Flinstone' StoryOfBedrock.txt

In another simple alias example, instead of always typing this ls command to get a directory listing:
ls -al
You may create alias so I only have to type the lowercase letter "L" like this:
l
Whenever I use this alias, it's exactly the same as if I had typed out the longer ls -al Linux command.
Using aliases like this you can create anything from simple shortcuts like this to powerfu custom commands.

Creating a Linux alias is very easy.

You can either enter them at the command line as you're working, or more likely, you'll put them in one of your startup files, like your .bashrc file, so they will be available every time you log in.

To create alias above by entering the following command into .bashrc file:
alias l="ls -al"
As you can see, the Linux alias syntax is very easy:

More examples

```
alias l="ls -al"
alias lm="ls -al|more"
alias html="cd /web/apache/htdocs/devdaily/html"
alias logs="cd /web/apache/htdocs/devdaily/logs"
alias qp="ps auxwww|more"
alias nu="who|wc -l"
alias aug="ls -al|grep Aug|grep -v 2008"
```

Unalias

The unalias command is used to remove entries from the current user's list of aliases.

Aliases can be created by using the alias command or by making entries in the appropriate configuration files.

unalias command is also built into several of the most commonly used shells, including ash, bash (the default shell on most Linux systems), csh and ksh.
The syntax of unalias is:

unalias [-a] [alias_name(s)]

For example, if a user had an alias named p for the pwd (i.e., present working directory) command, such alias could be removed with the following:

unalias p

unalias removes aliases created during the current login session.
It also suppresses permanent aliases; however, they are affected only for the current login session and are restored after the user logs in again.