BCA 3y

2

Linux Commands

Objectives:

After the completion of this chapter, you should know,

- How to interact with the system
- Directory oriented commands
- File oriented commands
- Process oriented commands
- Communication oriented commands
- Miscellaneous commands

COMMAND FORMAT

A command is an instruction given to the Shell; the Kernel will obey that instruction. Linux provides several commands for its users to easily work with it.

The general format of a command is,

```
command -options command_arguments
```

A command is normally entered in a line by typing from the keyboard. Even though the terminal's line width is 80 characters, the command length may exceed to 80 characters. The command simply overflows to the next line, though it is still in a single logical line.

Commands, *options* and *command_arguments* must be separated by white space(s) or tab(s) to enable the system to interpret them as words. *Options* must be preceded by a minus sign (-) to distinguish them from *command_arguments*. Moreover, options can be combined with only one minus sign.

For example, you can use the command, "`wc -l -w -c a.c`" as "`wc -lwc a.c`".

The command along with its *options*, *command_arguments* is entered in one line. This line is referred as "**command line**". A command line usually ends with a new-line character. The command line completes only after the user has hit the *[Enter]* key. The \ symbol placed at the end of a line continues the command to the next line, ignoring the hit of *[Enter]* key.

Several commands may be written in a single command line. They must be separated by semicolon (;).

For example, `$ date ; who`

The important Linux commands are grouped according to their functions and explained as follows.

- Directory Oriented Commands
- File Oriented Commands

- Process Oriented Commands
- Communication Oriented Commands
- General Purpose Commands
- Pipes and Filters

ECTORY ORIENTED COMMANDS

This command is used to list the content of the specified directory.

General format is,

```
ls [-options]  <directory_name>
```

where *options* can be,

a Lists all directory entries including the hidden files

l Lists the files in long format (filenames along with file type, file permissions, number of links, owner of the file, file size, file creation/modification time, number of links for a file). The number of links for a file refers more than one name for a file, does not mean that there are more copies of that file. This *ls -l* option displays also the year only when the file was last modified more than a year back. Otherwise, it only displays the date without year.

r Lists the files in the reverse order

t Lists the files sorted by the last modification time

R Recursively lists all the files and sub-directories as well as the files in the sub-directories.

p Puts a slash after each directory

s Displays the number of storage blocks used by a file.

x Lists contents by lines instead of by columns in sorted order

F Marks executable files with * (i.e the file having executable permission to the user) and directories with /

<directory_name> specifies a name of the directory whose contents are to be displayed. the *<directory_name>* is not specified, then the contents of the current directory are displayed.

Examples

```
[bmi@kousar bmi] $ ls
bmi    desktop   maxsizefile.sh    test1.txt   text
c      java      polindrome.sh     test2.txt

[bmi@kousar bmi] $ ls - r
text   test1.txt   maxisizefile.sh    Desktop     bmi
test2.txt          polindrome.sh      java        c

[bmi$kousar bmi] $ ls - l
total 36
                                  10    15:36 bmi
                                        11:11 c
```

```
-rwxrw-r--  1 bmi  bmi                214 Jan 10  15:24 maxsizefile.sh
-rwxrw-r--  1 bmi  bmi                382 Jan 10  14:41 polindrome.sh
-rw-rw-r--  1 bmi  bmi                 75 Jan 13  14:35 test1.txt
-rw-rw-r--  1 bmi  bmi                397 Jan 13  14:36 test2.txt
drwxrwxr-x  2 bmi  bmi               4096 Jan 10  12:11 text
[bmi@kousar bmi] $ ls -t
Desktop        test1.txt    bmi              polindrome.sh text
text2.txt      java         maxsizefile.sh   c
[bmi@kousar bmi] $ ls -a
.              .bash_history   c               polindrome.sh      text
..             .bash_logout    Desktop         .polindrome.sh.swp .ty.swp
.a2.swp        .bash_profile   jave            .screenrc          .Xauthority
.a.swo         .bashrc         .kde            test1.txt
.a.swp         bmi             maxsizefile.sh  test2.txt
[bmi@kousar bmi] $ ls -s
total 36
4 bmi          4 Desktop       4 maxsizefile.sh   4 test1.txt    4 text
4 c            4 java          4 polindrome.sh    4 test2.txt
[bmi@kousar bmi] $ ls -p
bmi/           Desktop         maxsizefile.sh     test1.txt      text/
c/             java/           polindrome.sh      test2.txt
[bmi@kousar bmi] $ ls -F
bmi/           Desktop         maxsizefile.sh*    test1.txt      text/
c/             java/           polindrome.sh*     test2.txt
[bmi@kousar bmi] $ ls -lt
total 36
drwxr-xr-x     2 bmi     bmi        4096 Dec 11   2002 Desktop
-rw-rw-r--     1 bmi     bmi         397 Jan 13  14:36 test2.txt
-rw-rw-r--     1 bmi     bmi          75 Jan 13  14:35 test1.txt
drwxrwxr-x     3 bmi     bmi        4096 Jan 10  10:03 java
drwxrwxr-x     4 bmi     bmi        4096 Jan 10  15:36 bmi
-rwxrw-r--     1 bmi     bmi         214 Jan 10  15:24 maxsizefile.sh
-rwxrw-r--     1 bmi     bmi         382 Jan 10  14:41 polindrome.sh
drwxrwxr-x     2 bmi     bmi        4096 Jan 10  12:11 c
drwxrwxr-x     2 bmi     bmi        4096 Jan 10  12:11 text
```

WILD CARD CHARACTERS

'*' represents any number of characters.

'?' represents a single character.

For example,

```
$ ls pgm*
```

This command will list out all the file-names of the current directory, which are starting with "pgm". Note that the suffix to pgm* may be any number of characters.

This command will display all the filenames of the current directory, which are ending with "s". Note that the prefix to s may be any number of characters.

```
$ ls ?gms
```

This command will display four character filenames, which are ending with "gms" starting with any of the allowed character. Note that the prefix to gms is a single character.

"[]" represents a subset of related filenames. This can be used with range operator "-" to access a set of files. Multiple ranges must be separated by commas.

```
$ ls pgm[1-5]
```

This command will list only the files named, pgm1, pgm2, pgm3, pgm4, pgm5 if they exist in the current directory. Note that the [1-5] represents the range from 1 through 5.

Examples

```
[bmi@kousar bmi] $ ls test?.txt
text1.txt    text2.txt    test3.txt

[bmi@kousar bmi] $ ls test[1-2].txt
text1.txt    text2.txt
```

2. mkdir

This mkdir (make directory) command is used to make (create) new directories.

General format is,

```
mkdir [-p] <directory_name1> <directory_name2>
```

The option *-p* is used to create consequences of directories using a single *mkdir* command.

Examples

```
$ mkdir ibr
```

This command will create '*ibr*' a subdirectory of the current directory.

```
$ mkdir x x/y
```

This command will create *x* as a subdirectory of current working directory, *y* as subdirectory of *x*.

```
$ mkdir ibr/ib/i
```

This command will make a directory *i* as a subdirectory of *ibr/ib*, but the directory structure - *ibr/i* must exist.

```
$ mkdir -p ibr/ib/i
```

Then, for the current directory, a subdirectory named *ibr* is created. Then, for the directory *ibr*, a subdirectory named *ib* is created. After that, the subdirectory *i* is created as a subdirectory of the directory *ib*.

3. rmdir

This rmdir (remove directory) command is used to remove (delete) the specified directories. A directory should be empty before removing it.

General format is,

```
rmdir [-p] <directory_name1> <directory_name2>
```

Example

```
$ rmdir ibr
```

This command will remove the directory *ibr*, which is the subdirectory of the current directory.

```
$ rmdir ibr/ib/i
```

This command will remove the directory *i* only.

```
$ rmdir -p ibr/ib/i
```

This command will remove the directories *i*, *ib* and *ibr* consequently.

4. cd

This cd (change directory) command is used to change the current working directory to a specified directory.

General format is,

```
cd <directory_name>
```

Examples

```
$ cd  /home/ibr
```

Then, the directory */home/ibr* becomes as the current working directory.

```
$ cd ..
```

This command lets you bring the parent directory as current directory. Here, .. represents the parent directory.

5. pwd

This pwd (print working directory) command displays the full pathname for the current working directory.

General format is,

```
pwd
```

Example

```
$ pwd
/home/bmi
```

Your present working directory is */home/bmi*.

6. find

This command recursively examines the specified directory tree to look for files matching some file attributes, and then takes some specified action on those files.

General format is,

```
find <path_list> <selection_criteria> <action>
```

It recursively examines all files in the directories specified in *<path_list>* and then matches each file for *<selection_criteria>* (file attributes). Finally, it takes the specified *<action>* on those selected files.

The *selection_criteria* may be as follows,

`-name <filename>` Selects the file specified in *<filename>*. If wild-cards are used, then double quote *<filename>*

`-user <username>` Selects files owned by ...

`-size {+n / -n}`	Selects files that are greater than/less than "*n*" blocks. (Generally one block is 512 bytes)
`-mtime {n / +n / -n}`	Selects files that have been modified on exactly *n* days / more than *n* days / less than *n* days
`-mmin {n / +n / -n}`	Selects files that have been modified on exactly *n* minutes / more than *n* minutes / less than *n* minutes
`-atime {n / +n / -n}`	Selects files that have been accessed on exactly *n* days / more than *n* days / less than *n* days
`-amin {n / +n / -n}`	Selects files that have been accessed exactly *n* minutes / more than *n* minutes / less than *n* minutes

The *action* may be as follows,

`-print`	Displays the selected files on the screen
`-exec <command>`	Executes the specified Linux command ends with {}\;

If *<path_list>* and *<action>* are not specified, then *current directory* and `-print` are taken respectively as default arguments.

Examples

```
$ find /home/ibrahim -name "*.java" -print
```

This command will recursively displays all *.java* files that are stored in the directory `/home/ibrahim` including all its sub-directories.

```
$ find /home/ibrahim -mtime 5 -print
```

If the current date is 20-02-2003, then this command will display the files that have been modified on 15-02-2003.

```
$ find /home/ibrahim -mtime +5 -print
```

If the current date is 20-02-2003, then this command will display the files that have been modified before 15-02-2003.

```
$ find /home/ibrahim -mtime -5 -print
```

If the current date is 20-02-2003, then this command will display the files that have been modified after 15-02-2003.

Making changes on a file means "*modifying*". Opening / Modifying a file means "*accessing*".

7. du

This du (disk usage) command reports the disk spaces that are consumed by the files in a specified directory, including all its sub-directories.

General format is,

```
du [-options] [-directory_name>]
```

With no arguments, 'du' reports the disk space for the current directory. Normally the

```
/dev/hda9          1035660      7100      975952      1%   /home
/dev/hda11         1778840    742916      945560     44%   /usr
/dev/hda10         1317920     17028     1423784      2%   /var
[bmi@kousar bmi] $ df
Filesystem           Inodes   IUsed   IFree  IUse%  Mounted on
/dev/hda8            205088   17204  187884     9%   /
/dev/hda6              6024      25    5999     1%   /boot
/dev/hda9            131616    1571  130045     2%   /home
/dev/hda11           226240   45867  180373    21%   /usr
/dev/hda10           193152     501  192651     1%   /var
```

FILE ORIENTED COMMANDS

1. cat

This cat (catenated -concatenate) command is used to display the contents of the specified file(s).

General format is,

```
cat [-options] <filename1> [<filename2> ...]
```

where *options* can be,

s	Suppresses warning about non-existent files
d	Lists the sub-directory entries only
b	Numbers non-blank output lines
n	Numbers all output lines

Examples

```
$ cat a.c
```

This will display the contents of the file *a.c*.

```
$ cat a.c b.c
```

This will display the contents of the files, *a.c* and *b.c*, one by one.

This command can be used with redirection operator (>) to create new files.

General format is,

```
cat > filename
<Type the text>
^d  (press [ctrl+d] at the end)
```

Example

```
$ cat  > x.txt
Hi! This
is
a file. Press [^d]
```

Then, a file named *x.txt* is created in the current working directory with 3 lines content.

2. cp

This command is used to copy the content of one file into another. If the destination is an existing file

where *options* can be,

- a Displays counts for all files, not just directories
- b Displays sizes in bytes
- c Displays output along with grand total of all arguments
- k Displays the sizes in KiloBytes
- m Displays the sizes in MegaBytes

Examples

```
[bmi@kousar bmi] $ du
12          ./.kde/Autostart
16          ./.kde
24          ./Desktop
16          ./c
12          ./text
4           ./bmi/c
4           ./bmi/text
152         ./bmi
4           ./java/ss
16          ./java
380

[bmi@kousar bmi] $ du /home/bmi/text
12          /home/bmi/text
[bmi@kousar bmi] $ du -a /home/bmi/text
4           /home/bmi/text/x.txt
4           /home/bmi/text/y.txt
12          /home/bmi/text
```

f

This df (disk free) command reports the available free space on the mounted file sys-
(disks).

General format is,

```
df [ options]
```

where *options* can be,

Shows local file systems only
Displays the sizes in KiloBytes
Displays the sizes in MegaBytes
Reports free, used, and percentage of used i-nodes.

s command reports the free spaces in blocks. Generally, one block is 512 bytes. The
try indicates that up to the specified number of files can be created on the file

nples

kousar bmi] $ df
ystem

General format is,

```
cp [-options] <source-file> <destination-file>
```

where *options* can be,

- `i` Prompt before overwriting destination files
- `p` Preserve all information, including owner, group, permissions, and timestamps
- `R` Recursively copies files in all subdirectories

Example

```
$ cp a.c b.c
```

The content of *a.c* is copied in to *b.c*.

3. rm

This rm (remove) command is used to remove (delete) a file from the specified directory. To remove a file, you must have *write* permission for the directory that contains the file, but you need not have permission on the file itself. If you do not have *write* permission on the file, the system will prompt before removing.

General format is,

```
rm [-options] <filename>
```

where *options* can be,

- `r` Deletes all directories including the lower order directories. Recursively deletes entire contents of the specified directory and the directory itself
- `i` Prompts before deleting
- `f` Removes write-protected files also, without prompting

Examples

```
$ rm a.c
```

This command deletes the file – *a.c* from the current directory. (But current directory will still exist with other files.)

```
$ rm -f/usr/ibrahim
```

This command deletes all the files and subdirectories of the specified directory */usr/ibrahim*. Note that the directory '*ibrahim*' also will be deleted.

4. mv

This mv (move) command is used to rename the specified files / directories.

General format is,

```
mv <source>. <destination>
```

Note that to make move, the user must have both write and execute permissions on the source>:

Example

```
$mv a.c b.c
```

Then, the file *a.c* is renamed to *b.c*.

c

This command is used to display the number of lines, words and characters of information in the specified file.

General format is,

```
wc [-options] <filename>
```

where *options* can be,

l	Displays the number of lines in the file
w	Displays the number of words in the file
c	Displays the number of characters in the file

Examples

```
$ wc a.c
```

It displays the number of lines, words and characters in the file – *a.c.*

```
$ wc -l a.c
```

It displays the number of lines in the file – *a.c.*

6. ln

This ln (link) command is used to establish an additional filename to a specified file. It doesn't mean that creating more copies of the specified file.

General format is,

```
ln <filename> <additional_filename>
```

where, *<filename>* is the name of the file for which *<additional_filename>* is to be established. The additional file names can be located on any directory. Thus, Linux allows a file to have more than one name, and yet maintain a single copy in the disk. But, changes to one of these files are also reflected to the others. If you delete one filename using *rm* command, then the other link-names will still exist.

Examples

```
[bmi@kousar bmi] $ ls -l test1.txt
-rw-rw-r--      1 bmi          bmi            75 Jan 13 14:35 test1.txt
[bmi@kousar bmi] $ ln test1.txt test2.txt
[bmi@kousar bmi] $ ls -l test*.txt
-rw-rw-r--      2 bmi          bmi            75 Jan 13 14:35 test1.txt
-rw-rw-r--      2 bmi          bmi            75 Jan 13 14:35 test2.txt
```

Note that the number of links for *test1.txt* and *test2.txt* are converted to 2.

```
[bmi@kousar bmi] $ rm test1.txt
[bmi@kousar bmi] $ ls -l test*.txt
-rw-rw-r--      1 bmi          bmi            75 Jan 13 14:35 test2.txt
```

7. file

This command lists the general classification of a specified file. It lets you to know if the content of the specified file is *ASCII text, C program text, data, separate executable, empty* or others.

General format is,

```
file <filename>
```

General format is,

```
comm [-options] <filename1> <filename2>
```

where *options* can be,

1 Suppresses listing of column1
2 Suppresses listing of column2
3 Suppresses listing of column3

Examples

```
[bmi@kousar bmi] $ cat file1.txt
I am ibrahim,
What is your name?
[bmi@kousar bmi] $ cat file2.txt
I am ibrahim,
What are you doing?
[bmi@kousar bmi] $ comm file1.txt file2.txt
                    I am ibrahim,
                What are you doing?
What is your name?
```

This command displays the lines that are unique to *a.c* in first *column 1*, the lines that are unique to *b.c* in *column 2* and the lines that are common for *a.c* and *b.c* in *column 3*.

```
[bmi@kousar bmi] $ -23 file.txt file2.txt
What is your name?
```

This command displays only the lines unique to the file -*file1.txt*, other columns (*column 1* and *column2*) are suppressed.

```
[bmi@kousar bmi] $ comm -1 file1.txt file2.txt
            I am ibrahim,
What are you doing?
```

FILE ACCESS PERMISSIONS

Linux treats everything as files. There are three types of files in Linux as follows,

- Ordinary file
- Directory file
- Special file (Device file)

The ordinary files consist of a stream of data that are stored on some magnetic media. A directory does not contain any data, but keeps track of an account of all the files and subdirectories that it contains. Linux treats even physical devices as files. Such files are called special files.

There are three types of modes for accessing these files as follows,

- Read mode (r)
- Write mode (w)

Example

```
[bmi@kousar bmi] $ file test1.txt
test1.txt: ASCII text
[bmi@kousar bmi] $ file test2.txt
test2.txt: ASCII text, with escape sequences
[bmi@kousar bmi] $ file *
Destop:          directory
bmi:             directory
c:               directory
java:            directory
shell:           ASCII text
test1.txt:       ASCII text
test2.txt:       ASCII text, with escape sequences
test3.txt:       ASCII English text
text:            directory
```

Here, '*' indicates the content of the current directory.

8. cmp

This cmp (compare) command is used to compare two files.
General format is,

```
cmp <filename1> <filename2>
```

This command reports the first instants of differences between the specified files. That is, the two files are compared byte by byte, and the location of the first mismatch is echoed to the screen.

Examples

```
[bmi@kousar bmi] $ cat file1.txt
I am ibrahim,
What is your name?
[bmi@kousar bmi] $ cat file2.txt
I am ibrahim,
What are you doing?
[bmi@kousar bmi] $ cmp file1.txt file2.txt
file1.txt file2.txt differ: char 20, line 2
```

The file1.txt differs from file2.txt at 20th character, which occurs at the 2nd line.

9. comm

This comm (common) command uses two sorted files as arguments and reports what is common. It compares each line of the first file with its corresponding line in the second file. The output of this command is in three columns as follows,

column1 Contains lines unique to filename1
column2 Contains lines unique to filename2
column3 Contains lines common for both filename1 and filename2