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

DIRECTORY ORIENTED COMMANDS

1. ls

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. If 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 - 1
total 36
drwxrwxr-x 4 bmi  bmi                4096  Jan 10    15:36 bmi
drwxrwxr-x 2 bmi  bmi                4096  Jan 10    12:11 c
drwxr-xr-x 2 bmi  bmi                4096  Dec 11     2002 Desktop
drwxrwr-x  3 bmi  bmi                4096  Jan 13    10:03 java
```

```
-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 ith "**pgm**". Note that the suffix to **pgm** may be any number of characters.

```
$ ls *s
```

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>
```

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

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 `<username>`
`-type d`	Selects directories

`-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 (**d**isk **u**sage) command reports the disk spaces that are consumed by the files 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 disk space is printed in units of 1024 bytes, but this can be overridden.

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
```

8. df

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

General format is,

```
df [-options]
```

where *options* can be,

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

This command reports the free spaces in blocks. Generally, one block is 512 bytes. The *i-nodes* entry indicates that up to the specified number of files can be created on the file system.

Examples

```
[bmi@kousar bmi] $ df
Filesystem     1k-blocks      Used    Aailable  Use%  Mounted on
/dev/hda8        1612808     63632     1467248    5%  /
/dev/hda6          23302      3489       18610   16%  /boot
```

```
/dev/hda9          1035660      7100      975952     1%   /home
/dev/hda11         1778840    742916      945560    44%   /usr
/dev/hda10         1517920     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 (**cat**enated -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 cp (**copy**) command is used to copy the content of one file into another. If the destination is an existing file, the file is overwritten; if the destination is an existing directory, the file is copied into that directory.

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 (**rem**ove) 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 (**mov**e) 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*.

wc

This command is used to display the number of lines, words and characters of informa-h stored on 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 -1 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 -1 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 -1 test*.txt
-rw-rw-r--       1 bmi          bmi           75 Jan 13 14:35 test2.txt
```

'. file

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

General format is,

```
file <filename>
```

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 filenam1 and filename2

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 (*column1* 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 sub-directories 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**)
- Execute mode (**x**)

The user can access a file with the above modes only if he/she has the corresponding permission. If he/she has only read permission on a file, then he/she cannot write and execute that file.

Linux separates its users into three groups for security and convenience as follows,

- User (u)
- User group (g)
- Others (o)

The system administrator assigns necessary file permissions to the above users.

We can know the file permissions of the files using "`ls -l`" command. This command will list the directory contents in long format including file permissions. The file permission is displayed in 10 characters width as follows,

BIT POSITION:	1	234	567	8910
MEANING:	file or directory (if d, then directory if –, then file)	rwx permission to users	rwx permission to usergroup	rwx permission to others

r	Readable
w	Writeable
x	Executable
-	Denial of permission

If a file has "`-rwxrw-r--`" as file permission, then the file is not a directory and it can be readable, writeable & executable for its *owner* (user), readable & writeable for its *usergroup* and only readable for *others*.

10. chmod

This chmod (**ch**ange **mod**e) command is used to change the file permissions for an existing file. We can use any one of the following notations to change file permissions.

- Symbolic notation
- Octal notation

Symbolic mode:

General format is,

```
chmod user_symbols set/deny_symbol access_symbols <filename(s)>
```

where *user_symbols* can be,

u	User
g	User group
o	Others

where *set/deny_ symbol* can be,

+	Assign the permissions
-	Remove the permissions
=	Assign absolute permission

where *access_symbols* can be,

r	Readable
w	Writeable
x	Executable

Examples

```
$ chmod u+x file1
```

This command adds *execute* permission to the user for executing the file – *file1*.

```
$chmod g+x file1
```

This command assigns *execute* permission to *usergroup* to execute the file – *file1* in addition to the existing permissions of that file. Note that the file holds its old permissions other than the changed *execute* permission i.e., the already existing permissions are not removed by default.

```
$chmod ugo-rwx file2
```

This command removes *read*, *write* and *execute* permissions from the *user*, *usergroup* and *others* for the file – *file2*. So, the file – *file2* will not *read, written and executed* by *user, usergroup and others*.

More than one permission can also be set with a single *chmod* command. The multiple file permission expressions must be delimited by commas (,).

```
$ chmod u+r,g-x,o+rw file1
```

This command assigns *read* permission to *users* (u+r), removes *execute* permission from *group* (g-x) and sets *read* & *write* permissions to *others* (o+rw).

The following command assigns *read* permission and removes *write* & *execute* permissions to/from *user*, *user group* and *others* on *file1*.

```
$ chmod ugo+r,ugo-wx file1
```

The above operation can also be achieved by using the following command.

```
$chmod ugo=r file1
```

Unlike the + or – operators, the absolute assignment (=) assigns only those permissions that are specified along with it, and removes other permissions.

Octal notation:

This mode uses a three-digit number to change the file permissions. In this number, the first digit represents *user* permissions, the second digit represents *usergroup* permissions and the third digit represents *others* permissions.

General format is,

```
chmod three_digit_number <filename1> [<filename2> ...]
```

Digits and their meanings,

0	No permissions
4	Read
2	Write
1	Execute

We can also sum the numbers for mixing of permissions.

3	Write and Execute
5	Read and Execute
6	Read and Write
7	Read, Write and Execute

Example

```
$ chmod 740 file1
```

Then, the *user* can *read, write* and *execute* the file – *file1*, because **7** is set as first digit. The *usergroup* can *only read* the file – *file1*, because **4** is set as second digit. The *other* cannot *read*, write and execute the file – *file1*, because **0** is set as third digit.

11. chown

This chown (**change ownership**) command is used to change the owner of a specified file. Only the owner of the file and the Superuser can change the file ownership.

General format is,

```
chown <new_owner> <filename>
```

This command changes <*new_owner*> as owner of the file specified in <*filename*>.

Example

```
[root@kousar bmi] # ls -l file1.txt
-rw-rw-r--       1 bmi          bmi            33 Jan 13 16:05 file1.txt
[root@kousar bmi] # chown ibrahim file1.txt
[root@kousar bmi] $ ls -l file1.txt
-rw-rw-r--       1 ibrahim      bmi            33 Jan 13 16:05 file1.txt
```

12. chgrp

This chgrp (**change group**) command is used to change the group ownership of a specified file. Only the owner of the file and the Superuser can change the group ownership of the file, irrespective of whether the user belongs to the same group or not.

General format is,

```
chgrp <new_groupname> <filename>
```

This command makes <*new_groupname*> as group-owner of the file specified in <*filename*>. Note that the group-owner of the file is also the *group* to which the file-owner belongs (no changes in default setting).

Example

```
[root@kousar bmi] # ls -l file1.txt
-rw-rw-r--       1 bmi          bmi            33 Jun 6 14:57 file1.txt
[root@kousar bmi] # chgrp mca file1.txt
[root@kousar bmi] $ ls -l file1.txt
-rw-rw-r--       1 bmi          mca            33 Jun 6 14:57 file1.txt
```

13. touch

This command is used to change the last modification and access time of a specified file in to the specified time.

General format is,

```
touch MMDDHHmm <filename1> [<filename2> ...]
```

where,

MM Month (01 -12)

DD Day (01 - 31)

HH Hour (00 - 23)

mm Minute (00 - 59)

If **MMDDHHmm** expression is not used, then the current date and time are taken by default.

Example

```
[bmi@kousar bmi] $ ls -l text1.txt
-rw-rw-r--       1 bmi          bmi            102 Jan 2 10:56 test1.txt
[bmi@kousar bmi] $ date
Thu Jan 13 16:55:28 IST 2000
[bmi@kousar bmi] $ touch test1.txt

[bmi@kousar bmi] $ ls -l test1.txt
-rw-rw-r--       1 bmi          bmi            102 Jan 13 16:55 test1.txt
```

The above *touch* command changed the last modified time and last access time of the file – *file1.txt* to current date and time.

```
[bmi@kousar bmi] $ touch 01100925 test1.txt
[bmi@kousar bmi] $ ls -1 test1.txt
-rw-rw-r--       1 bmi          bmi            102 Jan 10 09:25 test1.txt
```

The following command changes the last modified and access time of all the files in the current directory into the current date and time. Here, '*' represents all the files in the current directory.

```
[bmi@kousar bmi] $ touch *
```

14. dd

This command copies files and converts them in one format into another.

General format is,

```
dd [options=values]
```

where *options* can be,

`if` Input filename

`of` Output filename

`conv` File conversion specification. More than one conversion may be specified by separating them with commas.

The value for this option may be as follows,

`lcase` Converts uppercase letters into lowercase

`ucase` Converts lowercase letters into uppercase

`ascii` Converts the file by translating the character set from EBCDIC to ASCII

`ebcdic` Converts the file by translating the character set from ASCII to EBCDIC

Examples

```
[bmi@kousar bmi] $ cat small.txt
This is a file named SMALL.TXT
```

```
[bmi@kousar bmi] $ dd if="small.txt" of="capital.txt" conv=ucase
0+1 records in
0+1 records out
[bmi@kousar bmi] $ cat capital.txt
THIS IS A FILE NAMED SMALL.TXT
[bmi@kousar bmi] $ dd if="capital.txt" of="lower.txt" conv=lcase
0+1 records in
0+1 records out
[bmi@kousar bmi] $ cat lower.txt
this is a file named small.txt
```

15. expand

This command converts all the tabs present in the specified file into blank spaces a displays the result on the screen.

General format is,

```
expand [-i] <filename>
```

where the `-i` option converts only the initial tab into blank spaces.

Example

```
[bmi@kousar bmi] $ cat test.txt
This      is      a       file    named      test.txt
[bmi@kousar bmi] $ expand test.txt
This      is      a       file    named      test.txt
[bmi@kousar bmi] $ cat test.txt > test1.txt
[bmi@kousar bmi] $ ls -l test.txt test1.txt
-rw-rw-r--      1 bmi          bmi            49 Jan 21 09:36 test1.txt
-rw-rw-r--      1 bmi          bmi            30 Jan 21 09:35 test.txt
```

Note that the expanded file-size (*test1.txt*) is greater than the original file-size (*test.tx* because all the tabs (5) in the *test.txt* file are converted into blank spaces.

16. nl

This command numbers all non-blank lines in the specified text file and displays t same on the screen.

General format is,

```
nl <filename>
```

Example

```
[bmi@kousar bmi] $ cat test.txt
This is first line.

This is second line.


This is third line.
```

```
[bmi@kousar bmi] $ nl test.txt
     1    This is first line.

     2    This is second line.


     3    This is third line.
```

17. tac

This command reverses a file, so that the last line becomes the first line.

General format is,

```
tac <file_name>
```

Example

```
[bmi@kousar bmi] $ cat test.txt
This is the first line.
This is the second line.
This is the third line.

[bmi@kousar bmi] $ tac test.txt
This is the third line.
This is the second line.
This is the first line.
```

18. tail

This command displays the end of the specified file.

General format is,

```
tail ±n <filename>
```

If "**+n**" option is used, then the command will display from the n^{th} line to the end of the specified file. If "**-n**" option is used, then the last "n" lines of the specified file are displayed. If no option is specified, then the last 10 lines are displayed.

Examples

```
[bmi@kousar bmi] $ tail +4 employee.dat
1001        ismail       botany       bot        28-03-1965
1004        kadar        computer     cs         23-05-1988
[bmi@kousar bmi] $ tail -3 employee.dat
1003        abdulla      commerce     com        25-11-1985
1001        ismail       botany       bot        28-03-1965
1004        kadar        computer     cs         23-05-1988
[bmi@kousar bmi] $ ls -l | tail
-rwxrwxrwx     1 bmi      bmi            11 Dec 14 09:40 pgm1.sh
-rw-rw-r--     1 bmi      bmi           579 Dec 18 12:34 s1
-rw-rw-r--     1 bmi      bmi           282 Dec 19 12:41 ss
-rw-rw-r--     1 bmi      bmi         21153 Dec 19 09:21 t
-rw-rw-r--     1 bmi      bmi        211530 Dec 19 09:24 t1
-rw-rw-r--     1 bmi      bmi            31 Dec 16 09:23 test1.txt
```

```
-rw-rw-r--    1 bmi      bmi            31 Dec 14 09:39 test2.txt
-rw-rw-r--    1 bmi      bmi          1484 Dec 16 15:42 typescript
-rw-rw-r--    1 bmi      bmi            93 Dec 16 16:14 userlist.txt
-rw-rw-r--    1 bmi      bmi            46 Dec 19 12:18 www
```

This command displays the last 10 directory listing of the current directory.

19. head

This command displays the top of the specified file.

General format is,

```
head [-n] <filename>
```

If **-n** option is specified, then the first *n* lines of the file are displayed. Default value for this option is 10.

Examples

```
[bmi@kousar bmi] $ head -3 employee.dat
1005        yasin          computer       cs          15-08-1978
1002        abdulla        zoology        zoo         22-07-1956
1003        abdulla        commerce       com         25-11-1985
[bmi@kousar bmi] $ ls -1 | head
total 428
-rw-rw-r--    1 bmi      bmi         34520 Dec 24 10:28 a
-rw-rw-r--    1 bmi      bmi         16481 Dec 24 10:34 a1
-rw-rw-r--    1 bmi      bmi         19941 Dec 24 10:34 a2
drwxrwxr-x    2 bmi      bmi          4096 Dec 16 09:23 cpp
-rwxr----     1 bmi      bmi            21 Dec 16 09:54 d
drwxr-xr-x    2 bmi      bmi          4096 Dec 11 11:08 Desktop
-r--r--r--    1 bmi      bmi            21 Dec 11 09:54 dq
-rw-rw-r--    1 bmi      bmi           213 Dec 18 12:29 e
-rw-rw-r--    1 bmi      bmi            25 Dec 19 12:07 e1.dat
```

This command displays the first 10 directory listings of the current directory.

PROCESS ORIENTED COMMANDS

A process is a job in execution. Since Linux is a multi-user operating system, there might be several programs of several users running in memory. There is a program called *"Scheduler"* always running in memory which decides which process should get the CPU time and when. Note that only one process will be executed at a time, because *the system has only one processor (CPU).*

Some of the process-oriented commands are given below.

1. ps

This command is used to know which processes are running at our terminal.

General format is,

```
ps
```

ps -a: This command lists the processes of all the users who are logged on the system.

ps -t <terminal_name>: This command lists the processes, which are running on the specified terminal -*<terminal_name>*.

ps -u <user_name>: This command lists the processes, which are running for the specified user -*<username>*.

ps -x: This command lists the system processes. Apart from the processes that are generated by user, there are some processes that keep on running all the time. These processes are called *system processes*.

Examples

```
$ ps
$ ps -a
$ ps -t tty3d
```

This displays the processes, which are running on the terminal - *tty3d*.

```
$ ps -u ibrahim
```

This displays the processes, which are running for the user - *ibrahim*.

BACKGROUND PROCESSING

Linux provides the facility for background processing. That is, when one process is running in the foreground, another process can be executed in the background. The ampersand (&) symbol placed at the end of a command sends the command for background processing.

Example

```
$ sort emp.doc&
```

By the execution of this command, a number is displayed. This is called **PID** (***P**rocess **ID**entification*) number. In Linux, each and every process has a unique PID to identify the process. The PIDs can range from **0** to **32767**.

Then, the command '*sort emp.doc*' will run on background. We can execute another command in foreground (as normal).

2. kill

If you want a command to terminate prematurely, press *[ctrl+c]*. This type of interrupt characters does not affect the background processes, because the background processes are protected by the Shell from these interrupt signals. This *kill* command is used to terminate a background process.

General format is,

```
kill [-SignalNumber] <PID>
```

The PID is the process identification number of the process that we want to terminate. Use *ps* command to know the PIDs of the current processes.

By default, this *kill* command uses the signal number 15 to terminate a process. But, some programs like login shell simply ignore this signal of interruption, and continue execution normally. In this case, you can use the signal number 9 (often referred as *sure kill*).

Examples

```
$kill 120
```

This command terminate the process who has the PID 120.

```
$kill -9 130
$kill -9 0
```

This command kills all the processes including the login Shell. (The Kernel itself being the first process gets the PID 0. The above command kills the Kernel, so all the processes are killed.)

3. nohup

If a user wants a process that he has executed not to die even when he logged-out from the system, you can use this nohup (**no hangup**) command. Note that normally all the processes of a user are terminated if he logs out.

General format is,

```
nohup <command>&
```

Example

```
$ nohup sort emp.doc&
1116
```

Here, 1116 is the PID of the process. Then, the command *sort emp.doc* will be executed even if you logged out from the system. The output of this command will be stored in a file named *nohup.out*.

4. at

This command is used to execute the specified Linux commands at future time.

General format is,

```
at <time>
<commands>
^d
```

(Press *[ctrl + d]* at the end)

Here, *<time>* specifies the time at which the specified *<commands>* are to be executed.

Example

```
$ at 12:00
echo "LUNCH BREAK"
^d
```

at offers the keywords – ***now, noon, midnight, today*** and ***tomorrow*** and they convey special meanings.

```
$ at noon                          ← 12:00
echo "LUNCH BREAK"
^d
```

at also offers the keywords ***hours, days, weeks, months*** and ***years***, can be used with + operator as shown in the following examples.

```
$ at 12:00 + 1 day                 ← at 12:00 tomorrow
$ at 13:00 Jan 20, 2003 + 2 days   ← at 1 pm January 20, 2004
```

The **atq** command is used to list the jobs submitted by you on at queue. This command lists the *job number, scheduled date of execution.*

General format is,

```
atq
```

The **atrm** command is used to remove a job from *at* queue.

General format is,

```
atrm <jobnumber>
```

Example

```
$ atq
7    2002-12-16 12:00 a bmi
8    2002-12-17 12:00 a bmi
9    2003-01-22 13:00 a bmi

$ atrm 8

$ atq
7    2002-12-16 12:00 a bmi
9    2003-01-22 13:00 a bmi
```

5. batch

This command is used to execute the specified commands when the system load permits (when CPU becomes nearly free).

General format is,

```
batch
<commands>
^d
```

Any job scheduled with batch also goes to the *at* queue, you can list or delete them through *atq* and *atrm* respectively.

Example

```
$ batch
sort a.c
sort b.c
^d
```

COMMUNICATION ORIENTED COMMANDS

Linux provides the communication facility, from which a user can communicate with the other users. The communication can be ***online*** or ***offline***. In online communication, the user, to whom the message is to be sent (recipient), must be logged on the system. In offline communication, the recipient need not be logged on the system.

Some of the communication-oriented commands are given below:

1. write

This online communication command lets you to write messages on another user's terminal.

General format is,

```
write <RecipientLoginName>
<message>
^d
```

Example

```
$ wirte ibrahim
Hello ibrahim!
How are you?
^d
```

On the execution of this command, the specified message is displayed on the terminal of the user - *ibrahim*.

If the recipient does not want to allow these messages (sent by other users) on his terminal, then he can use "**mesg n**" command. If he wants to revoke this option and want to allow any one to communicate with him, then he can use "**mesg y**" command.

General format is,

```
mesg [y|n]
```

y Allows *write* access to your terminal.

n Disallows *write* access to your terminal.

The ***mesg*** without argument give the status of the mesg setting.

The "**finger**" command can be used to know the users who are currently logged on the system and to know which terminals of the users are set to *mesg y* and which are set to *mesg n*. A '*' symbol is placed on those terminals where the ***mesg*** set to ***n***.

2. mail

This command offers *off-line* communication.

General format is,

To send mail,

```
mail <username>
<message>
^d
```

The mail program mails the message to the specified user. If the user (recipient) is logged on the system, the message "*you have new mail*" is displayed on the recipient's terminal. However, the user is logged on the system or not, the mail will be kept in the mailbox until the user issues the necessary command to read the mails.

To check mails, give the ***mail*** command without arguments. This command will list all the incoming mails received since the latest usage of the mail command. A & symbol is displayed at the bottom. This is called **mail prompt**. Here we can issue several mail prompt commands (referred as internal commands).

Some commands which can be given in mail prompt are given below,

Mail Prompt Commands	Functions
+	Displays the next mail message if exists
-	Displays the previous mail message if exists
<number>	Displays the <number>th mail message if exists
D	Deletes currently viewed mail and displays next mail message if exists
d <number>	Deletes the <number>th mail
s <filename>	Stores the current mail message to the file specified in <filename>
s<number> <filename>	Stores the <number>th mail message to the file specified in <filename>
R	Replies to the sender of the currently viewing mail
r <number>	Replies the <number>th mail to its sender
Q	Quits the mail program

3. wall

Usually, this wall (**write all**) command is used by the super-user to send a message to all the users who were currently logged on the system.

General format is,

```
wall
<message>
<Press [ctrl + d] at the end>
```

Example

```
$ wall
Meeting at 16:00 hrs.
^d
```

The specified message "*Meeting at 16:00 hrs.*" will be displayed on everyone's terminal with a beep sound like ***write***'s message, but ignoring *mesg* settings.

GENERAL PURPOSE COMMANDS

1. date

This command displays the system's date and time.

General format is,

```
date +<format>
```

where <*format*> can be,

%H	Hour – 00 to 23
%I	Hour – 00 to 12
%M	Minute – 00 to 59
%S	Second – 00 to 59
%D	Date – MM/DD/YY

%T	Time – HH:MM:SS
%w	Day of the week
%r	Time in AM / PM
%y	Last two digits of the year

Examples

```
$ date
Mon Dec 16 15:13:10 IST 2002

$ date +%H
15

$ date +%I
03

$ date +%M
13

$ date +%S
10

$ date +%D
12/16/02

$ date +%T
15:13:10

$ date +%w
1          ← 0=Sunday, 1=Monday, 2=Tuesday, ...

$ date +%r
03:13:10 PM

$ date +%y
02
```

2. who

Since Linux is a multi-user operating system, several users may work on this system. This command is used to display the users who are logged on the system currently.

General format is,

who

Example

```
$ who
ibrahim     tty1     Feb 19 10:17
sheik       tty3     Feb 19 10:20
mukash      tty8     Feb 19 11:02
```

The first column of the output represents the user names. The second column represents the corresponding terminal names and the remaining columns represents the time at which the users are logged on.

3. who am I

This command tells you who you are. (Working on the current terminal)

General format is,

```
who am i
```

Example

```
$ who am i
ibrahim  tty1     Feb    19  10:17
```

4. man

This man (**manual**) command displays the syntax and detailed usage of the Linux command, which is supplied as argument.

General format is,

```
man <LinuxCommand>
```

Example

```
$ man wc
```

This will display the help details for "*wc*" command.

Almost all of the commands offer **--help** option that displays a short listing of all the options.

```
$ wc --help
```

5. cal

This command will display calendar for the specified month and year.

General format is,

```
cal [<month>] <year>
```

where, month can be ranged from 1 to 12.

Example

```
$ cal 2002
```

This command will display calendar for the year 2002 (for 12 months).

```
$ cal 1 2003
```

This will display the calendar for the month - January of the year 2003.

```
$ cal 2 2003
    February 2003
Su Mo Tu We Th Fr Sa
                   1
 2  3  4  5  6  7  8
 9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28
```

6. lpr

This command is used to print one or more files on printer.

General format is,

```
lpr [-options] <filename> <filename2> ... <filenameN>
```

where *options* can be,

r Removes file(s) from directory after printing

m Mail inform you when printing is over

Example

```
$ lpr a.c
```

7. tee

This command does the operations of *pipe* and *redirection*. It will send the output of a command into standard output as well to a specified file.

General format is,

```
command | tee <filename>
```

Examples

```
$ ls
$ ls > list.doc
```

We can combine these two commands in to a single command (using *tee*) as follows,

```
$ ls | tee list.doc
```

The output of the command *'ls'* is sent to the standard output device (screen) and also to the file – *list.doc*.

```
$ who | tee userlist.txt | wc -l
```

This command stores the current user information in to the file named *userlist.doc* and displays the number of current users.

The **-a** option can be used with the *tee* command, which appends the output of the specified command into the specified file. Otherwise, the file content will be overwritten.

```
$ cat a.txt
This is a file named a.txt

$ cat b.txt
This is a file named b.txt

$ cat a.txt | tee result.txt
This is a file named a.txt

$ cat result.txt
This is a file named a.txt

$ cat b.txt | tee result.txt
This is a file named b.txt

$ cat result.txt
This is a file named b.txt
```

```
$ cat a.txt | tee -a result.txt
This is a file named a.txt

$ cat result.txt
This is a file named b.txt
This is a file named a.txt
```

8. script

This command stores your login session in to a specified file. All the operations that you have done (all the given commands, their outputs, error messages, ...) are stored on that file.

Start the login session by issuing the following command.

```
$ script
```

Then do your operations, and finally use the **exit** command to end script. Then all the operations in between *script* and *exit* will be stored on a file named ***typescript***.

The next script command will overwrite the *typescript* file. You can append instead of overwriting by using the *script* command with **-a** option.

```
$ script -a
```

Otherwise you can use other file-name instead of the default *-typescript* by specifying the filename as argument.

```
$ script actions.txt
```

Example

```
[bmi@kousar bmi] $ script actions.txt
Script started, file is actions.txt
[bmi@kousar bmi] $ cat test.txt
This is a test file.
[bmi@kousar bmi] $ wc emp.doc
     20         18         51 emp.doc
[bmi@kousar bmi] $ exit                    ← Press [Enter]
```

Then, the content of *actions.txt* is,

```
Script started on Fri Jun  6 16:21:20 2003
[bmi@kousar bmi]$ cat test.txt
This is a test file.
bmi@kousar bmi]$ wc emp.doc
20       18       51 emp.doc
[bmi@kousar bmi]$ exit
Script done on Fri Jun  6 16:21:44 2003
```

9. tput

This command with ***clear*** option can be used to clear the screen content.

General format is,

```
tput clear
```

You can also use **clear** command for this purpose.

Example

```
$ tput clear
```

10. split

If a file is very large, then it cannot be edited on an editor. In such situations, we need to split the file in to several small files. For this purpose, this split command is used.

General format is,

```
split -<number> <filename>
```

The `-<number>` option splits the file specified in *<filename>* into *<number>* lined files. Default for this option is 1000 lines.

The splitted contents are stored in the file names xaa, xab, xac, ... ,xaz, xba, xbb, xbc, ... ,xzz (totally 676 filenames).

Example

```
$ split test.txt
```

If the specified file - *test.txt* contains 5550 lines, then this command creates the files namely *xaa, xab, xac, xad, xae* containing 1000 lines and xaf containg remaining 550 lines.

```
$ split -500 test.txt
```

This command splits the test.txt file into 500 lined files.

11. expr

This command is used to perform arithmetic operations on integers.

The arithmetic operators and their corresponding functions are given below.

+	Addition
-	Subtraction
*	Multiplication
/	Division (Decimal portion will be truncated. Because it perform division operation on integers only. It gives only quotient of the division.)
%	Remainder of division (modulus operator)

A white space must be used on either side of an operator. Note that the multiplication operator (*) has to be escaped to prevent the Shell from interpreting it as the filename meta-character. This command only works with integers, so, the division yields only integer part.

Examples

```
$ x=5
$ y=2
```

Then,

```
$ expr $x + $y
7

$ expr $x - $y
3

$ expr $x \* $y          ← Escape the multiplication operator
10
```

```
$ expr $x / $y
2                    ← Note that decimal portion is truncated.
$ expr $x % $y
1                    ← Remainder part of division.
$ expr $x + 10
15
```

12. bc

This command is used to perform arithmetic operations on integers as well as on floats (decimal numbers).

Type the arithmetic expression in a line and press *[Enter]* key. Then the answer will be displayed on the next line. After you have finished your work, press *[Ctrl + d]* keys to end up.

Examples

Add 10 and 20.

```
$ bc
10 + 20    ← arithmetic expression
30         ← result is displayed on the next line
```

Press ^d to end the work.

Divide 8 by 3.

```
$ bc
8 / 3
2                    ← Decimal portion is truncated.

$ bc
a=8
b=3
c=a/b
c
2
```

By default, *bc* performs truncated division (integer division). If you do not want to truncate any value (requiring float division), then you have to set **scale** to the number of digits of precision before the operation.

```
$ bc
scale=1
8/3
2.6
scale=2
8/3
2.66

^d
```

Note that the result (2.66) is not rounded off; the actual result is 2.6666...

This bc command can be used with **ibase** (input base) and **obase** (output base) to convert numbers in one bash into another.

```
$ bc
ibase=2     ← Set ibase to binary (2)
101         ← Type the input binary number
5           ← Result in decimal

$ bc
obase=2     ← Set obase to binary
5           ← Type the input decimal number
101         ← Result in binary

$bc
ibase=16    ← Set ibase to hexadecimal
B           ← Type the input hexadecimal number
11          ← Result in decimal

$bc
obase=16    ← Set obase to hexadecimal
11          ← Type the input decimal number
B           ← Result in hexa-decimal
```

SUMMARY

All the necessary commands, which are used to interact with the Linux system, are grouped and explained.

The three file access modes are Read (r), Write (w) and Execute (e). The three groups of Linux users are: user (u), user group (g) and others (o). Their behaviours are explained in detail.

A process is a job in execution. The process-oriented commands are explained separately

Linux provides online as well as offline communication facility. In online communication the recipient must be logged with the system during communication process. But, in offline communication, the recipient need not to be logged with the system. The communication-oriented commands are explained with examples. And the frequently used commands are illustrated with suitable examples.

SELF-ASSESSMENT QUESTIONS

1. How are the Linux directory structures organized? Explain the directory access commands.
2. How do you compare the contents of two files? Explain.
3. How do you grant and revoke permissions to Linux files and directories? Explain with suitable examples.
4. Write a short note on Background processing in Linux.
5. Explain the Linux support of offline communication.
6. How do you restrict the messages, which are sent from other terminals through write command?
7. Explain the commands that are used for arithmetic operations