3

Pipes and Filters

Objectives:

After the completion of this chapter, you should know,

- Uses of Pipe
- Redirection procedure
- Commands used for filtering data and output.
- The workingness of *sed* and *gawk* utilities

INTRODUCTION

PIPE

"Pipe" is a mechanism in which the output of one command can be redirected as input to another command.

General format is,

```
command1 | command2
```

The output of *command1* is sent to *command2* as input.

Example

```
$ ls | more
```

The output of the command *'ls'* is sent to the *'more'* command as input. So, the directory listing of the current directory is displayed page by page (with pause).

The following two-command group displays the total number of users who are currently loged on the system.

```
$ who > userlist.txt
$ wc -l userlist.txt
```

Note that unnecessarily a file named *userlist.txt* is created. If we use pipe mechanism, there is no need to create such temporary file.

```
$ who | wc -l
```

REDIRECTION

Linux treats the keyboard as the standard input (value 0) and terminal screen as standard output (value 1) as well as standard error (value 2). However, input can be taken from sources other than the keyboard and output can be passed to any source other than the terminal screen. Such a process is called ***"redirection"***.

Redirecting inputs:

The '<' symbol is used to redirect inputs.

Example

```
$ cat < file1.txt
```

Then the file – *file1.txt* is taken as input for the command –*cat*.

Redirecting outputs:

The ">" symbol is used to redirect outputs.

Example

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

Then, the output of the command '*ls*' is stored on the file '*list.doc*'. We can also use '1>' instead of '>'.

Redirecting Error messages:

The '2>' symbol is used to redirect error messages.

Example

```
$ cat list1.doc
```

If there is no file named '*list1.doc*' in the current directory, then the error message is sent to the standard error device (usually screen). We can redirect this error message using '2>' symbol.

```
$ cat list1.doc 2> errormes.txt
```

Then, the error message, if generated, will be stored on the file - *errormes.txt*.

FILTERS

There are some Linux commands that accept input from standard input or files, perform some manipulation on it, and produces some output to the standard output. Since these commands perform some filtering operations on data, they are appropriately called as "*filters*". These filters are used to display the contents of a file in sorted order, extract the lines of a specified file that contains a specific pattern, etc.

1. sort

This command sorts the contents of a given file based on ASCII values of characters.

General format is,

```
sort [options] <filename>
```

where *options* can be,

`-m <filelist>`	Merges sorted files specified in <filelist>
`-o <filename>`	Stores output in the specified <filename>
`-r`	Sorts the content in reverse order (reverse alphabetical order)
`-u`	Removes duplicate lines and display sorted content
`-n`	Numeric sort
`-t"char"`	Uses the specified "*char*" as delimiter to identify fields
`-c`	Checks if the file is sorted or not
`+pos`	Starts sort after skipping the posth field

-pos	Stops sorting after the posth field
+pos.n	Starts sort after the nth column of the (pos+1)th field
-pos.n	Stops sort on the nth column of the (pos+1)th field

Example

```
$ cat empname.txt
yasin
abdulla
ibrahim
abdulla
ismail

$ sort empname.txt
abdulla
abdulla
ibrahim
ismail
kadar
yasin
```

This command displays sorted contents of the file *-empname.txt* on the screen.

```
$ sort -r empname.txt
yasin
kadar
ismail
ibrahim
abdulla
abdulla
```

This command displays the sorted content, sorted in reverse alphabetical order, of th file *–empname.txt*.

```
$ sort empname.txt -o result.txt

$ cat result.txt
abdulla
abdulla
ibrahim
ismail
kadar
yasin
```

This command stores the sorted contents of "*empname.txt*" in to "*result.txt*".

```
$ cat empname1.txt
yasin
abdulla
ibrahim
```

```
$ cat empname2.txt
ismail
kadar
abdulla

$ sort empname1.txt > e1.txt
$ sort empname2.txt > e2.txt

$ cat e1.txt
abdulla
ibrahim
yasin

$ cat e2.txt
abdulla
ismail
kadar

$ sort -m e1.txt e2.txt
abdulla
abdulla
ibrahim
ismail
kadar
yasin
```

This command merges the sorted contents of the files *-e1.txt* and *e2.txt*. It is always preferable to sort files separately before merging them.

```
$ sort empname.txt
abdulla
abdulla
ibrahim
ismail
kadar
yasin

$ sort -u empname.txt
abdulla
ibrahim
ismail
kadar
yasin
```

The sort with -u option removes duplicate lines from output.

Generally, sorting takes place in the order of numerals, uppercase letters and then lower case letters. As the rule, the digits, alphabets and other special characters are converted to their ASCII (*American Standard Code for Information Interchange*) value. Then, this sort arranges its input according to their ASCII value. Since sorting is based on ASCII values of the characters, the line *"20"* is less than the line *"3"*(for example). So, the result will be unexpectable. The sort with **-n** option will arrange its input according to numerical values.

```
$ cat numbers.txt
5
10
7
20

$ sort numbers.txt
10
20
5
7

$ sort -n numbers.txt
5
7
10
20
```

The *sort* with -c option is used to check whether the specified file has been sorted not. If the file is sorted, no message will be displayed, otherwise it will indicate the unsortedne

```
$ sort -c numbers.txt
sort: numbers.txt:2: disorder: 10

$ sort numbers.txt > n.txt
$ sort -c n.txt
$
```

If a file contains records (written in a single line) containing fields normally separ by a blank space or a tab space, and if we want to sort the file based on any particular f then +pos & -pos options are used.

```
[bmi@fayas bmi] $ cat employee.dat
1005        yasin        computer        cs        15-08-1978
1002        abdulla      zoology         zoo       22-07-1956
1006        ibrahim      computer        cs        18-09-1987
1003        abdulla      commerce        com       25-11-1985
1001        ismail       botany          bot       28-03-1965
1004        kadar        computer        cs        23-05-1988
```

(Records are separated by a tab space)

```
[bmi@fayas bmi] $ sort employee.dat
1001        ismail      botany      bot     28-03-1965
1002        abdulla     zoology     zoo     22-07-1956
1003        abdulla     commerce    com     25-11-1985
1004        kadar       computer    cs      23-05-1988
1005        yasin       computer    cs      15-08-1978
1006        ibrahim     computer    cs      18-09-1987
[bmi@fayas bmi] $ sort +1 employee.dat
1003        abdulla     commerce    com     25-11-1985
1002        abdulla     zoology     zoo     22-07-1956
1006        ibrahim     computer    cs      18-09-1987
1001        ismail      botany      bot     28-03-1965
1004        kadar       computer    cs      23-05-1988
1005        yasin       computer    cs      15-08-1978
```

The argument **+1** indicates that sorting should start after skipping the first field. So, the above sort command does sorting after skipping the first field (employee number field). If you want to start from third field, use **+2** as argument.

```
[bmi@fayas bmi] $ sort +2 employee.dat
1001        ismail      botany      bot     28-03-1965
1003        abdulla     commerce    com     25-11-1985
1005        yasin       computer    cs      15-08-1978
1006        ibrahim     computer    cs      18-09-1987
1004        kadar       computer    cs      23-05-1988
1002        abdulla     zoology     zoo     22-07-1956
```

The argument +1 indicates that sorting should start after skipping the first field. And the argument -2 indicates that sorting should **stop** after the second field. The following command sorts the *employee.dat* according to empname (second field). Note that this output differs from the output of "***sort +1 employee.dat***".

```
[bmi@fayas bmi] $ sort +1 -2 employee.dat
1002        abdulla     zoology     zoo     22-07-1956
1003        abdulla     commerce    com     25-11-1985
1006        ibrahim     computer    cs      18-09-1987
1001        ismail      botany      bot     28-03-1965
1004        kadar       computer    cs      23-05-1988
1005        yasin       computer    cs      15-08-1978
```

The default field separator for sort command is a blank space or a tab space. You can specify any other character as field separator by using **-t** option.

```
[bmi@fayas bmi] $ cat employee.dat
1005|yasin|computer|cs|15-08-1978
1002|abdulla|zoology|zoo|22-07-1956
1006|ibrahim|computer|cs|18-09-1987
1003|abdulla|commerce|com|25-11-1985
1001|ismail|botany|bot|28-03-1965
1004|kadar|computer|cs|23-05-1988
```

```
[bmi@fayas bmi] $ sort employee.dat
1001|ismail|botany|bot|28-03-1965
1002|abdulla|zoology|zoo|22-07-1956
1003|abdulla|commerce|com|25-11-1985
1004|kadar|computer|cs|23-05-1988
1005|yasin|computer|cs|15-08-1978
1006|ibrahim|computer|cs|18-09-1987
[bmi@fayas bmi] $ sort -t"|" +1 -2 employee.dat
1002|abdulla|zoology|zoo|22-07-1956
1003|abdulla|commerce|com|25-11-1985
1006|ibrahim|computer|cs|18-09-1987
1001|ismail|botany|bot|28-03-1965
1004|kadar|computer|cs|23-05-1988
1005|yasin|computer|cs|15-08-1978
```

The above command treats the character "|" as field separator.

If you want to sort fourth field (dept. code) as primary key and second field (empname) as secondary key, use the following *sort* command.

```
[bmi@fayas bmi] $ cat employee.dat
1005        yasin        computer        cs        15-08-1978
1002        abdulla      zoology         zoo       22-07-1956
1006        ibrahim      computer        cs        18-09-1987
1003        abdulla      commerce        com       25-11-1985
1001        ismail       botany          bot       28-03-1965
1004        kadar        computer        cs        23-05-1988
[bmi@fayas bmi] $ sort +3 -4 +1 employee.dat
1001        ismail       botany          bot       28-03-1965
1003        abdulla      commerce        com       25-11-1985
1006        ibrahim      computer        cs        18-09-1987
1004        kadar        computer        cs        23-05-1988
1005        yasin        computer        cs        15-08-1978
1002        abdulla      zoology         zoo       22-07-1956
```

The argument +3 indicates that the sort should start after the third field and should stop after the fourth field (i.e., exactly fourth field). The third argument +1 indicates that resumption of sort starts after the first field.

You can also specify the character position within a specified field to be the beginning and ending positions of sort. The following command will sort the file *employee.dat* according to last four digits of the fifth field (year -after the fourth field, in which after the sixth character).

```
[bmi@fayas bmi] $ sort +4.6 employee.dat
1002        abdulla      zoology         zoo       22-07-1956
1001        ismail       botany          bot       28-03-1965
1005        yasin        computer        cs        15-08-1978
1003        abdulla      commerce        com       25-11-1985
1006        ibrahim      computer        cs        18-09-1987
1004        kadar        computer        cs        23-05-1988
```

The following command will sort the file *employee.dat* according to first two digits of the fifth field (date).

```
[bmi@fayas bmi] $ sort +4.0 -4.3 employee.dat
1005        yasin       computer      cs      15-08-1978
1006        ibrahim     computer      cs      18-09-1987
1002        abdulla     zoology       zoo     22-07-1956
1004        kadar       computer      cs      23-05-1988
1003        abdulla     commerce      com     25-11-1985
1001        ismail      botany        bot     28-03-1965
```

2. grep

This grep (global search for regular expression) command is used to search for a specified pattern from a specified file and display those lines containing the pattern.

General format is,

```
grep [-options] pattern <filename>
```

(Quote the pattern if the pattern contains Shell special characters.)

where *options* can be,

b	Ignores spaces, tabs
i	Ignores case distinction for matching (do not differentiate capital and small case letters)
v	Displays only the lines that do not match the specified pattern
c	Displays the total number of occurrences of the pattern in the file
n	Displays the resultant lines along with their line numbers
<number>	Displays the matching lines along with <number> of lines above and below

Example

```
[bmi@kousar bmi] $ cat employee.dat
1005        yasin       computer      cs      15-08-1978
1002        abdulla     zoology       zoo     22-07-1956
1006        ibrahim     computer      cs      18-09-1987
1003        abdulla     commerce      com     25-11-1985
1001        ismail      botany        bot     28-03-1965
1004        kadar       computer      cs      23-05-1988
[bmi@kousar bmi] $ grep "cs" employee.dat
1005        yasin       computer      cs      15-08-1978
1006        ibrahim     computer      cs      18-09-1987
1004        kadar       computer      cs      23-05-1988
[bmi@kousar bmi] $ grep -l "ibrahim" employee.dat
1002        abdulla     zoology       zoo     22-07-1956
1006        ibrahim     computer      cs      18-09-1987
1003        abdulla     commerce      com     25-11-1985
```

REGULAR EXPRESSION CHARACTER SET

*	Matches zero or more characters
.	Matches a single character (equalent to ? in Shell)

[r1-r2]	Matches a single character within the ASCII range represented by the characters
[^abcd]	Matches a single character which is not a, b, c or d
^<character>	Matches the lines that are beginning with the character specified in <character>
<character>$	Matches the lines that are ending with the character specified in <character>

```
[bmi@kousar bmi] $ grep "com*" employee.dat
1005        yasin          computer        cs        15-08-1978
1006        ibrahim        computer        cs        18-09-1987
1003        abdulla        commerce        com       25-11-1985
1004        kadar          computer        cs        23-05-1988
[bmi@kousar bmi] $ grep "8$" employee.dat
1005        yasin          computer        cs        15-08-1978
1004        kadar          computer        cs        23-05-1988
[bmi@kousar bmi] $ ls -l | grep "^d"
drwxrwxr-x   2 bmi          bmi             4096 Dec 16 09:23 cpp
drwxr-xr-x   2 bmi          bmi             4096 Dec 11 11:08 Desktop
```

This command displays only directory entries. Note that directory entries are starting with "d".

egrep

The egrep (extended global search for regular expression) command offers additional features than grep. Multiple patterns can be searched by using pipe symbol (|).

Example

```
[bmi@kousar bmi] $ grep "ibrahim|smail" employee.dat
                                                    ← No Result
[bmi@kousar bmi] $ egrep "ibrahim|smail" employee.dat
1006        ibrahim        computer        cs        18-09-1987
1001        ismail         botany          bot       28-03-1965
```

fgrep

This fgrep (**fixed grep**) command works like grep, but it does not accept *regular expressions* unlike *grep*.

Examples

```
[bmi@kousar bmi] $ fgrep "cs" employee.dat
1005        yasin          computer        cs        15-08-1978
1006        ibrahim        computer        cs        18-09-1987
1004        kadar          computer        cs        23-05-1988
[bmi@kousar bmi] $ fgrep "c*" employee.dat
[bmi@kousar bmi] $
```

3. uniq

This uniq (**unique**) command is used to handle duplicate lines in a file. If this command is used without any option, it displays the lines by eliminating duplicate lines.

General format is,

```
uniq [-option] <filename>
```

where *options* can be,

u	Displays only the non-repeated lines
d	Displays only the duplicated lines
c	Displays each line by eliminating duplicate lines, and prefixing the number of times it occurs.

Examples

```
[bmi@kousar bmi] $ cat employee.dat
1005        yasin       computer    cs      15-08-1978
1002        abdulla     zoology     zoo     22-07-1956
1002        abdulla     zoology     zoo     22-07-1956
1006        ibrahim     computer    cs      18-09-1987
1006        ibrahim     computer    cs      18-09-1987
1006        ibrahim     computer    cs      18-09-1987
1003        abdulla     commerce    com     25-11-1985
1001        ismail      botany      bot     28-03-1965
1004        kadar       computer    cs      23-05-1988
[bmi@kousar bmi] $ uniq employee.dat
1005        yasin       computer    cs      15-08-1978
1002        abdulla     zoology     zoo     22-07-1956
1006        ibrahim     computer    cs      18-09-1987
1003        abdulla     commerce    com     25-11-1985
1001        ismail      botany      bot     28-03-1965
1004        kadar       computer    cs      23-05-1988
[bmi@kousar bmi] $ uniq -u employee.dat
1005        yasin       computer    cs      15-08-1978
1003        abdulla     commerce    com     25-11-1985
1001        ismail      botany      bot     28-03-1965
1004        kadar       computer    cs      23-05-1988
[bmi@kousar bmi] $ uniq -d employee.dat
1002        abdulla     zoology     zoo     22-07-1956
1006        ibrahim     computer    cs      18-09-1987
[bmi@kousar bmi] $ uniq -c employee.dat
   1 1005        yasin       computer    cs      15-08-1978
   2 1002        abdulla     zoology     zoo     22-07-1956
   3 1006        ibrahim     computer    cs      18-09-1987
   1 1003        abdulla     commerce    com     25-11-1985
   1 1001        ismail      botany      bot     28-03-1965
   1 1004        kadar       computer    cs      23-05-1988
```

4. more

If the information to be displayed on the screen is very long, it scrolls up on the screen fastly. So, the user cannot be able to read it. This *more* command is used to display the output page by page (without scrolling up on the screen fastly). Use *[spacebar]* or *[f]* key to scroll forward one screen, use *[b]* key to scroll backward one screen, use *[q]* key to quit displaying.

General format is,

```
more <filename>
```

Example

```
$ more a.c
```

This command will display the content of the file –*a.c* page by page.

```
$ ls | more
```

This command will display the directory listing page by page.

You can also use more specialized command **-less**, instead of *more*. This *less* comman has the same syntax and functions of *more*.

```
$ less a.c

$ ls | less
```

Note: For this paging purpose, you can also use key combinations. The *[ctrl+s]* is used to st scrolling of the screen output. The *[ctrl+q]* is used to resume scrolling of the screen output. But it is an ideal method.

5. pr

This command displays the contents of the specified file adding with suitable head and footers. This command can be used with *lpr* command for neat hard copies. The Hea part consists of the last modification date and time along with file-name and page numbe

General format is,

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

where *options* can be,

`-l <number>`	It changes the page size to specified <*number*> of lines (By default, page size is 66 lines)
`-<number>`	Prepares the output in <*number*> columns
`-n`	Numbers lines
`-t`	Turns off the heading at the top of the page

Examples

```
[bmi@kousar bmi] $ cat employee.dat
1005        yasin        computer        cs        15-08-1978
1002        abdulla      zoology         zoo       22-07-1956
1003        abdulla      commerce        com       25-11-1985
1001        ismail       botany          bot       28-03-1965
1004        kadar        computer        cs        23-05-1988
[bmi@kousar bmi] $ pr employee.dat
2002-12-19 10:13            employee.dat                Page    1


1005        yasin        computer        cs        15-08-1978
1002        abdulla      zoology         zoo       22-07-1956
1003        abdulla      commerce        com       25-11-1985
1001        ismail       botany          bot       28-03-1965
1004        kadar        computer        cs        23-05-1988
```

—Note that remaining lines are empty—

```
[bmi@kousar bmi] $ pr -n employee.dat                    Page  1

2002-12-19 10:13                 employee.dat

1     1005      yasin      computer      cs      15-08-1978
2     1002      abdulla    zoology       zoo     22-07-1956
3     1003      abdulla    commerce      com     25-11-1985
4     1001      ismail     botany        bot     28-03-1965
5     1004      kadar      computer      cs      23-05-1988
```

—Note that remaining lines are empty—

```
[bmi@kousar bmi] $ pr -2 employee.dat                    Page  1

2002-12-19 10:13                 employee.dat

1005      yasin      computer      cs      15-08-1978
1002      abdulla    zoology       zoo     22-07-1956
1003      abdulla    commerce      com     25-11-1985
```

—Note that remaining lines are empty—

6. cut

This command is used to cut the columns / fields of a specified file (Like the head and tail commands cut the lines - rows).

General format is,

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

where options can be,

`c <columns>` Cuts the columns specified in *<columns>*. You must separate the column numbers by using commas

`f <fields>` Cuts the fields specified in *<fields>*. You must separate field numbers by using commas

Examples

```
[bmi@kousar bmi] $ cat employee.dat
1005      yasin      computer      cs      15-08-1978
1002      abdulla    zoology       zoo     22-07-1956
1003      abdulla    commerce      com     25-11-1985
1001      ismail     botany        bot     28-03-1965
1004      kadar      computer      cs      23-05-1988
[bmi@kousar bmi] $ cut -f 3 employee.dat
computer
zoology
commerce
botany
computer
[bmi@kousar bmi] $ cut -f 3-5 employee.dat
computer      cs      15-08-1978
zoology       zoo     22-07-1956
commerce      com     25-11-1985
botany        bot     28-03-1965
computer      cs      23-05-1988
```

```
[bmi@kousar bmi] $ cut -f 1-3,5 employee.dat
1005          yasin           computer        15-08-1978
1002          abdulla         zoology         22-07-1956
1003          abdulla         commerce        25-11-1985
1001          ismail          botany          28-03-1965
1004          kadar           computer        23-05-1988
[bmi@kousar bmi] $ cat e.dat
1005|yasin|computer|cs|15-08-1978
1002|abdulla|zoology|zoo|22-07-1956
1003|abdulla|commerce|com|25-11-1985
1001|ismail|botany|bot|28-03-1965
1004|kadar|computer|cs|23-05-1988
[bmi@kousar bmi] $ cut -d'|' -f 3-4 e.dat
computer|cs
zoology|zoo
commerce|com
botany|bot
computer|cs
[bmi@kousar bmi] $ cut -c 1-4 e.dat
1005
1002
1003
1001
1004
[bmi@kousar bmi] $ cut -c 6-10,15,17 e.dat
yasinpt
abdulol
abdulom
ismaitn
kadarpt
```

7. paste

This command concatenates the contents of the specified files into a single file vertically. (Like *cut* command separates the columns, this *paste* command merges the columns).

General format is,

```
paste <filename1> <filename2> ...
```

Examples

```
[bmi@kousar bmi] $ cat e1.dat
1005
1002
1003
1001
1004
```

```
[bmi@kousar bmi] $ cat e2.dat
computer
zoology
commerce
botany
computer
[bmi@kousar bmi] $ paste e1.dat e2.dat
1005            computer
1002            zoology
1003            commerce
1001            botany
1004            computer
```

8. tr

This tr (**translate**) command is used to change the case of alphabets.

General format is,

```
tr <CharacterSet1> <CharacterSet2> <StandardInput>
```

This command translates the first character in the *<CharacterSet1>* into the first character in the *<CharacterSet2>* and this same procedure is continued for remaining characters. Note that this command gets input from ***standard input***, not from a file. But you can use pipe or redirection to use a file as input.

Examples

```
[bmi@kousar bmi] $ cat e2.dat
computer
zoology
commerce
botany
computer
[bmi@kousar bmi] $ cat e2.dat | tr "[a-z]" "[A-Z]"
COMPUTER
ZOOLOGY
COMMERCE
BOTANY
COMPUTER
[bmi@kousar bmi] $ tr "c,o" "C,O" < e2.dat
COmputer
zOOlOgy
COmmerCe
bOtany
COmputer
[bmi@kousar bmi] $ tr -d "c,o" < e2.dat
mputer
zlgy
mmerce
btany
mputer
```

4 VI Editor

Objectives:

After the completion of this chapter, you should know,

- How to work with vi editor
- The three modes of vi editor
- Insert, delete, replace text, cursor movement, search, yanking, redo and undo, search commands, etc.

INTRODUCTION

Linux offers various types of editors like **ex, sed, ed, vi, vim, xvi, nvi, elvis** etc., to create and edit your files (data files, program files, text files etc.). The famous one is ***vi*** editor (visual full screen editor) created by Bill Joy at the University of California at Berkeley.

STARTING VI

This editor can be invoked by typing **vi** at the **$** prompt. If you specify a filename as an argument to vi, then the vi will edit the specified file, if it exists.

```
vi [<filename>]
```

A status line at the bottom of the screen (25th line) shows the filename, current line and character position in the edited file.

```
vi +<linenumber> <filename>
```

- Edits the file specified in *<filename>* and places the cursor on the *<linenumber>*th line.

VI MODES

The vi editor works on *three modes* as follows:

INSERT MODE:

- The text should be entered in this mode. And any key press in this mode is treated as text.
- We can enter into this mode from command mode by pressing any of the keys:

  `i, I, a, A, o, O, r, R, s, S.`

COMMAND MODE:

- It is the default mode when we start up vi editor.
- All the commands on vi editor (cursor movement, text manipulation, etc.) should be used in this mode.

- We can enter into this mode from *Insert mode* by pressing the *[Esc]* key, and from *Ex mode* by pressing *[Enter]* key.

EX MODE:

- The ex mode commands (saving files, find, replace, etc.,) can be entered at the last line of the screen in this mode.
- We can enter into this mode from command mode by pressing *[:]* key.

Note that one cannot enter to *ex mode* directly from *input mode* and vice versa.

The following are some of the commands that should be used in *command mode*.

INSERT COMMANDS:

i	Inserts before cursor
I	Inserts at the beginning of the current line (the line at which the cursor is placed)
a	Appends after cursor
A	Appends at the end of the current line
o	Inserts a blank line below the current line
O	Insert a blank line above the current line

DELETE COMMANDS:

x	Deletes a character at the cursor position
<n>x	Deletes specified number (n) of characters from the cursor position
X	Deletes a character before the cursor position
<n>X	Deletes specified number (n) of characters before the cursor position
dw	Deletes from cursor position to end of the current word. It stops at any punctuation (eg. ' , .) that appears with the word
dW	Same as "dw"; but ignores any punctuation that appears with the word
db	Deletes from cursor position to beginning of the current word. It stops at any punctuation that appears with the word
dB	Same as "db"; but ignores any punctuation that appears with the word
dd	Deletes current line
<n>dd	Deletes specified number of lines (n) from the current line
d*[Enter]*	Deletes current line and the following line
d0	Deletes all the characters from the beginning of the current line to previous character of cursor position
D	Deletes all the characters from the current character to the end of the current line
d/<pattern>/*[Enter]*	Deletes all characters but before to specified ***pattern*** occurs

d)	Deletes all the characters from current cursor position to end of the current sentence
d(	Deletes all the characters from current cursor position to the beginning of the current sentence
d}	Deletes all the characters from the current cursor position to the end of the current paragraph
d{	Deletes all the characters before the cursor position to beginning of the current paragraph

REPLACE COMMANDS:

r	Replaces single character at the cursor position
R	Replaces characters until *[Esc]* key is pressed from current cursor position
s	Replaces single character at the cursor position with any number of characters
S	Replaces entire line

CURSOR MOVEMENT COMMANDS:

h or *[back space]*	Moves cursor to the left (left arrow)
l or *[space bar]*	Moves cursor to the right (right arrow)
k	Moves cursor to up (up arrow)
j	Moves cursor down (down arrow)
w	Forwards to first letter of next word; but stops at any punctuation that appears with the word
b	Backwards to first letter of previous word; but stops at any punctuation that appears with the word
e	Moves forward to the end of the current word; but stops at any punctuation that appears with the word
W	Same as w; but ignores punctuation that appears with the word
B	Same as b; but ignores punctuation that appears with the word
E	Same as e; but ignores punctuation that appears with the word
[Enter]	Forwards to beginning of next line
0	Moves to the first location of the current line
^	Moves to the first character of the current line

$	Moves to the last character of the current line
H	Moves to the first character (left end) of top line on the current screen
M	Moves to the first character (left end) of middle line on the current screen
L	Moves to the first character (left end) of lowest line on the current screen
G	Moves to the first character of the last line in the current file
<n>G	Moves to the first character of the specified line (**n**) in the current file
(	Moves to the first character of the current sentence
)	Moves to the first character of next sentence
{	Moves to the first character of current paragraph
}	Moves to the first character of next paragraph

SEARCH COMMANDS:

/string*[Enter]*	Searches the specified ***string*** forward in the file
?string*[Enter]*	Searches the specified ***string*** backward in the file
n	Finds the next string in the same direction (specified by the above commands)
N	Finds the next string in the opposite direction (specified by the above commands)

YANKING COMMANDS: (COPY & PASTE)

yy (or) Y	Yanks the current line in to the buffer (copy)
nyy (or) **n**Y	Copies the '**n**' lines from the current line to the buffer
p	Pastes the yanked text below the current line (below the cursor)
P	Pastes the yanked text above the current line (above the cursor)

REDO COMMAND:

. (period)	Repeats the most recent editing operation performed. Note that it is not applicable to cursor movement commands.

UNDO COMMAND:

u	Undoes the most recent editing operation performed.

For example, assume that you have deleted a line currently, if you use this *undo* command, then the deleted line will be displayed (undone the delete operation).

SCREEN COMMANDS: Forward

Ctrl F	Scrolls full screen (screen of text - page) forward
Ctrl B	Scrolls full screen backward
Ctrl D	Scrolls half screen forward
Ctrl U	Scrolls half screen backward
Ctrl G	Display the status (*filename, total number of lines in the file, current l number, percentage of file that precedes the cursor*) on the status line bottom of the screen)

Some of the ***ex*** mode commands are given below. These commands should be used mode prefixed by colon (:)

:w	Saves file without quitting
:w <filename>	Saves the content into a file specified in *<filename>*
:m,n w <filename>	Saves the lines *m* through *n* into the specified file
:.w <filename>	Saves the current line into the specified file
:$w <filename>	Saves the last line into the specified line
:x (or) :wq	Saves file and quits from *vi*
:q!	Quits from *vi* without saving
:sh	Escape to the Linux shell (Temporarily exits from *vi*, by typi ***exit*** or pressing ***[ctrl + d]*** on $ prompt, we can enter into that session.
:s/s1/s2	Does a single replacement s1 as s2 on text (text of the ed file). Press **n** for next match on the same direction or Press next match on the opposite direction
:s/s1/s2/g	Does the replacement s1 as s2 throughout the text
:n,m s/s1/s2/g	Does the replacement in between the lines n and m
:.s/s1/s2/g	Does the replacement only on the current line
:$s/s1/s2/g	Does the replacement only on the last line
:set number	Displays the line numbers sequentially. If we delete/inser then the line numbers are adjusted automatically
:set nonumber	Removes the line numbers, which are set by ***:set number*** command
:set showmode	Displays the currently working mode at the last line

:!<command>	Executes the specified command without exiting from *vi (not escaping to Linux Shell)*
:<linenumber>	Moves the cursor to the line specified in *<linenumber>*
:set tabstop=<no>	Sets the tab setting to *<no>* number of spaces. (Default is 8 spaces)
:set ignorecase	Ignores case while searching for patterns
:<n1>co<n2>	Copies the line <n1> into below of the line <n2>
:<n1>,<n2>co<n3>	Copies the lines <n1> through <n2> into below of the line <n3>
:<n1>m<n2>	Moves the line <n1> into the line <n2>
:<n1>,<n2>m<n3>	Moves the lines from <n1> through <n2> into the line <n3>
:<n1>d	Deletes the line <n1>
:<n1>,<n2>d	Deletes the lines <n1> through <n2>

SUMMARY

Vi works on three different modes.

The text should be inserted in the insert mode.

All the commands should be entered in command mode.

One can enter into the command mode from the Insert mode by pressing the [Esc] key, and from Ex mode by pressing [Enter] key.

The ex mode commands should be entered prefixed by colon (:).

SELF-ASSESSMENT QUESTIONS

1. Write a note on different modes of *vi* editor.
2. Explain ex mode commands.
3. How do you insert and delete text in *vi* editor?
4. List and illustrate cursor movement commands.