MODULE 3. SIMPLE FILTERS

3.1 INTRODUCTION

Filters are the commands which accept data from standard input, manipulate it and write the results to standard output. Filters are very important tools of the UNIX system. Many UNIX files have lines containing *fields*. Some commands expect these fields to be separated by a suitable delimiter, which is not a part of the data. The delimiter may be a space, colon (:), comma (,), pipe (|) etc. In the current study, we will use a file *emp.lst* as a database in which fields are delimited by pipe (|).

```
$cat emp.lst
2233|a.k. shukla|g.m.|sales|12/12/52|6000
9876|jai sharma|director|production|03/12/50|7000
5678|sumit chakrobarty|d.g.m|marketing|04/19/43|6000
2365|barun sengupta|director|personnel|05/11/47|7800
5423|n.k. gupta|chairman|admin|08/30/56|5400
1006|chanchal sanghvi|director|sales|09/03/38|6700
6213|karuna ganguly|g.m.|accounts|06/05/62|6300
1265|s.n. dasgupta|manager|sales|09/12/63|5600
4290|jayant Chodhury|executive|production|09/07/50|6000
2476|anil aggarwal|manager|sales|05/01/59|5000
6521|lalit chowdury|director|marketing|09/26/45|8200
3212|shyam saksena|d.g.m|accounts|12/12/55|6000
3564|sudhir Agarwal|executive|personnel|07/06/47|8000
2345|j.b. saxena|g.m.|marketing|03/12/45|8000
0110|v.k. agrawal |g.m.|marketing|12/31/40|9000
```

The above file is used in all the examples of this chapter. In this file, the fields are EmpID, Name, Designation, Department, Date of Birth and Salary.

3.2 pr: PAGINATING FILES

The ***pr*** command prepares a file for printing by adding suitable headers, footers and formatted text. The command uses filename as an argument as below –

```
$pr emp.lst

May 06 10:38 1997   emp.lst              Page1

2233|a.k. shukla|g.m.|sales|12/12/52|6000
9876|jai sharma|director|production|03/12/50|7000
5678|sumit chakrobarty|d.g.m|marketing|04/19/43|6000
2365|barun sengupta|director|personnel|05/11/47|7800
5423|n.k. gupta|chairman|admin|08/30/56|5400
1006|chanchal sanghvi|director|sales|09/03/38|6700
…………..
………………
(full file will be displayed)
```

Observe that a header includes the date and time of last modification, name of the file and Page number. The ***pr*** command adds five lines of margin at the top and five lines at the bottom.

NOTE: When you run the above command, if you cannot see the file contents, it means that the contents of file have been scrolled up. So, you may see the blank lines only. To avoid this, use the ***more*** option as below –

```
$ pr emp.lst |more
```

Now, only few lines at a time will be displayed and you can keep pressing enter – key to read fully.

3.2.1 pr Options

The ***pr*** command uses several options – few of which are listed in Table 3.1.

Option	Meaning
-t	Suppresses head and footer
-k (where k is an integer)	Prints the file contents in *k* columns
-d	Double-line spacing between the lines of file contents
-n	Assigns numbers to lines. It is useful in debugging the code
-o *n*	Offsets lines by *n* spaces and increases left margin of page.

Let us discuss some of these options applied on the output of ***cat emp.lst***.

Ex1.

```
$ pr –t -3 emp.lst
2233|a.k. shuk   9876|jai sharm   5678|sumit cha
2365|barun sen   5423|n.k. gupt   1006|chanchal
6213|karuna ga   1265|s.n. dasg …………..
………………………………… (full file will be printed in 3 columns)
```

As the option `–t` is used, the header part of the `pr` command is suppressed. The option `-3` indicates that the contents have to be displayed in 3 columns on a standard output device (monitor). But observe that, each record in the given file is quite big and cannot fit into 3 columns. Hence, some characters from each record are omitted so as to fit 3 columns in the output display.

Ex2.

```
$ pr –t –d -3 emp.lst
2233|a.k. shuk   9876|jai sharm   5678|sumit cha

2365|barun sen   5423|n.k. gupt   1006|chanchal

6213|karuna ga   1265|s.n. dasg …………..
………………………………… (full file will be printed in 3 columns)
```

The output is displayed in 3 columns by giving double-line spacing between the lines.

Ex3.
```
     $ pr -t -n emp.lst
     1  2233|a.k. shukla|g.m.|sales|12/12/52|6000
     2  9876|jai sharma|director|production|03/12/50|7000
     3  5678|sumit chakrobarty|d.g.m|marketing|04/19/43|6000
     ………..
     ……………….
    15  0110|v.k. agrawal |g.m.|marketing|12/31/40|9000
```

Each line of the file is numbered here.

Ex4.
```
$ pr -t -o 5 emp.lst
     2233|a.k. shukla|g.m.|sales|12/12/52|6000
     9876|jai sharma|director|production|03/12/50|7000
     5678|sumit chakrobarty|d.g.m|marketing|04/19/43|6000
     ……..
```
Here, the contents of the files are displayed by leaving 5 spaces as the left margin.

Note that any combination of various options given in Table 3.1 can be used to combine the effects.

3.3 head: DISPLAYING THE BEGINNING OF A FILE

The ***head*** command is used to display top of the file. By default, it displays first 10 lines of the file. The `-n` option can be used with a required line–count to display those many lines. For example,
```
$ head -n 3 emp.lst
2233|a.k. shukla|g.m.|sales|12/12/52|6000
9876|jai sharma|director|production|03/12/50|7000
5678|sumit chakrobarty|d.g.m|marketing|04/19/43|6000
```

Here, `-n` is used with the count 3; hence the top 3 lines of the file will be displayed.

3.4 tail: DISPLAYING THE END OF A FILE

The ***tail*** command is opposite to ***head.*** That is, it displays last 10 lines by default. By specifying the count with –n option, we can display only the required number of lines as below –
```
$ tail -n 3 emp.lst
3564|sudhir Agarwal|executive|personnel|07/06/47|8000
2345|j.b. saxena|g.m.|marketing|03/12/45|8000
0110|v.k. agrawal |g.m.|marketing|12/31/40|9000
```

The ***tail*** command can be used to display lines from a specific line number as below –
```
$ tail -n +14 emp.lst
2345|j.b. saxena|g.m.|marketing|03/12/45|8000
0110|v.k. agrawal |g.m.|marketing|12/31/40|9000
```

Here, from 14th line, till the last line output is displayed. As there are 15 lines in the file emp.lst, the above command has displayed two lines.

3.4.1 tail Options

There are various options for ***tail*** command. We will discuss two of them here.

- **Monitoring File Growth (-f):** Many UNIX programs constantly write to the log files of the system as long as they are running. System admin need to monitor the growth of these files to view the latest messages. The –f (follow) option of ***tail*** is used for this purpose. The following example is used to monitor the installation of Oracle by watching the growth of the log file *install.log* from another terminal.
```
$tail –f  /oracle/app/oracle/product/11.1/orainst/install.log
```

  The prompt doesn't return even after the work is over, and hence one has to use the interrupt key (like Ctrl +C) to abort the process.

- **Extracting Bytes Rather than Lines (-c):** The –c option with ***tail*** is used to extract required number of bytes from the file.

  Ex1.
```
$ tail -c -100 emp.lst
 |8000
 2345|j.b. saxena|g.m.|marketing|03/12/45|8000
 0110|v.k. agrawal |g.m.|marketing|12/31/40|9000
```

  Here, the last 100 bytes from the file emp.lst are extracted.

  Ex2.
```
$ tail -c +500 emp.lst
 lit chowdury|director|marketing|09/26/45|8200
 3212|shyam saksena|d.g.m|accounts|12/12/55|6000
 3564|sudhir Agarwal|executive|personnel|07/06/47|8000
 2345|j.b. saxena|g.m.|marketing|03/12/45|8000
 0110|v.k. agrawal |g.m.|marketing|12/31/40|9000
```

  Here, the data is extracted after skipping first 500 bytes.

3.5 cut: SLITTING A FILE VERTICALLY

The ***cut*** command is used in two types:

- **Cutting Columns (-c):** To extract specific columns, the ***–c*** option is used with a list of column numbers delimited by comma. One can give range of columns using hyphen. The following example extracts the columns from 6th to 22nd and again from 24th to 32nd from the file emp.lst.
```
$ cut -c 6-22,24-32 emp.lst
a.k. shukla|g.m.|ales|12/1
jai sharma|directr|product
sumit chakrobartyd.g.m|mar
barun sengupta|diector|per
…………………………………………………….
```

Observe that there is no space before/after comma when you are specifying two lists of columns.

- **Cutting Fields (-f):** The ***–c*** option is useful for fixed length lines (or fields). But we cannot expect it to happen for all the files (for example, name of different individuals cannot contain equal number of characters). Hence, extracting (or cutting) fields makes sense rather than extracting characters. The ***–f*** option makes it possible with a default delimiter as tab. If the delimiter is any other character than tab, ***–d*** option should be used. Following is an example to extract 2nd and 3rd fields from the file *emp.lst.*

```
$ cut -d \| -f 2,3 emp.lst
a.k. shukla|g.m.
jai sharma|director
sumit chakrobarty|d.g.m
barun sengupta|director
……….
```

Note that in the file *emp.lst* the delimiter is **|** (pipe). But, UNIX system uses **|** symbol as a pipeline character which is normally used to make output of one command to be an input for another command. Hence, to avoid UNIX system from misunderstanding the **|** as pipeline character, we need to use it as an escape sequence `\|`.

An alternative way of providing delimiter character is - within double quotes. In the following example, the 1st, 4th, 5th and 6th columns are extracted from *emp.lst.*

```
$ cut -d "|" -f 1,4- emp.lst
2233|sales|12/12/52|6000
9876|production|03/12/50|7000
5678|marketing|04/19/43|6000
2365|personnel|05/11/47|7800
…………………
```

The field specification `1,4-` given here indicates 1st field, and 4th field onwards. We need not specify `1,4,5,6` here. That is, when series of fields required till the last field, the hyphen will do the job.

3.6 paste: PASTING FILES

The ***paste*** command is used to paste the contents into a file vertically. One can view two files side by side by pasting them. To understand the working of *paste* command, first let us create two files by cutting some fields from *emp.lst.*

The file *cutlist1* is created as below, which contains 2nd and 3rd fields (name and designation) of the file *emp.lst*.

```
$ cut -d \| -f 2,3 emp.lst > cutlist1
$ cat cutlist1
a.k. shukla|g.m.
jai sharma|director
sumit chakrobarty|d.g.m
barun sengupta|director
……………………………………………
```

The file *cutlist2* is created as below, which contains 1st and 4th to last fields of the file *emp.lst*.

```
$ cut -d "|" -f 1,4- emp.lst > cutlist2
$ cat cutlist2
2233|sales|12/12/52|6000
9876|production|03/12/50|7000
5678|marketing|04/19/43|6000
2365|personnel|05/11/47|7800
……………………………………………………
```

Now, use ***paste*** command to paste two files *cutlist1* and *cutlist2* vertically as below –

```
$ paste cutlist1 cutlist2
a.k. shukla|g.m.        2233|sales|12/12/52|6000
jai sharma|director     9876|production|03/12/50|7000
sumit chakrobarty|d.g.m 5678|marketing|04/19/43|6000
barun sengupta|director 2365|personnel|05/11/47|7800
……………………………………………………
```

The ***paste*** command uses tab as the delimiter between two files. If we wish to give the delimiter of our choice, -d option should be used. In the below given example, the comma (,) symbol is used as delimiter between two files.

```
$ paste -d "," cutlist1 cutlist2
a.k. shukla|g.m. , 2233|sales|12/12/52|6000
jai sharma|director , 9876|production|03/12/50|7000
sumit chakrobarty|d.g.m , 5678|marketing|04/19/43|6000
barun sengupta|director , 2365|personnel|05/11/47|7800
…………………………………………………………
```

Though we say that two files should be given as input to *paste* command, the data for one file can be given through standard input. For example, if the file *cutlist2* doesn't exist, provide the character stream by cutting the required fields and then piping it to paste command as below –

```
$ cut -d \| -f 1,4- emp.lst | paste -d "|" cutlist1 -
a.k. shukla|g.m.|2233|sales|12/12/52|6000
jai sharma|director|9876|production|03/12/50|7000
sumit chakrobarty|d.g.m|5678|marketing|04/19/43|6000
barun sengupta|director|2365|personnel|05/11/47|7800
……………………………………………………
```

Observe the hyphen after *cutlist1* in the above example. The order of pasting can be reversed by changing the position of hyphen as below –

```
$ cut -d \| -f 1,4- emp.lst | paste -d "|" - cutlist1
 2233|sales|12/12/52|6000| a.k. shukla|g.m.
 9876|production|03/12/50|7000| jai sharma|director
 5678|marketing|04/19/43|6000| sumit chakrobarty|d.g.m
 2365|personnel|05/11/47|7800| barun sengupta|director
  ……………………………………………………
```

The –s option of ***paste*** command can be used to join lines. For example, assume we have file *addressbook* as below –

```
$cat addressbook
Ajay                    # first person Name
ajay@gmail.com          #email
9120783220                   #phone
Vijay                        # second person name
vijay@yahoo.com
8034125789
```

Here, *name, email, phone* are in different lines, but actually represents single person. The whole file is having all fields in different lines, instead of one in-front of other. To combine these lines, we can use –s option as below –

```
$ paste –s addressbook
Ajay ajay@gmail.com 9120783220 Vijay vijay@yahoo.com 8034125789
```

Now, again this result is meaningless as all records are put in a single line. But, it would be better if we keep *name, email, phone* of every person in one line. For this purpose, we can use –d option along with –s option as below –

```
$ paste –s –d “||\n” addressbook
Ajay|ajay@gmail.com|9120783220
Vijay|vijay@yahoo.com|8034125789
```

Here, we have used `“||\n”` as delimiter. Here, two pipe symbols are acting as delimiter after joining two lines of the original file. The new-line character is a delimiter after joining 3rd line. In case we put 3 pipe symbols, the result would be different as below –

```
$ paste –s –d “|||\n” addressbook
Ajay|ajay@gmail.com|9120783220|Vijay
vijay@yahoo.com|8034125789
```

3.7 sort: ORDERING A FILE

Sorting is arranging data in ascending or descending order. By default, the ***sort*** command reorders the lines in ASCII collating sequence (white space first, then numerals, uppercase, lowercase). For example,

```
$ sort emp.lst
0110|v.k. agrawal |g.m.|marketing|12/31/40|9000
1006|chanchal sanghvi|director|sales|09/03/38|6700
1265|s.n. dasgupta|manager|sales|09/12/63|5600
………………………………………………………………………………………
………………………………………………………………………………………
9876|jai sharma|director|production|03/12/50|7000
```

3.7.1 Sorting Options

The important options of ***sort*** command are given in Table 3.2. Some of the options are discussed with examples here.

Table 3.2 Options of ***sort*** command

Option	Description
`-t`*`char`*	Uses delimiter *char* to identify fields
`-k` *`n`*	Sorts on nth field
`-k` *`m,n`*	Starts sort on mth field and ends sort on nth field
`-k` *`m.n`*	Starts sort on nth column of mth field
`-u`	Removes repeated lines
`-n`	Sorts numerically
`-r`	Reverses sort order
`-f`	Folds lowercase to equivalent uppercase (case-insensitive sort)
`-m` *`list`*	Merges sorted files in *list*
`-c`	Checks if file is sorted
`-o` *`fname`*	Places output in file *fname*

- **Sorting on Primary Key (-k):** Following is the example to sort the file *emp.lst* based on the 2nd field *name*.

```
$ sort -t "|" -k 2 emp.lst
2233|a.k. shukla|g.m.|sales|12/12/52|6000
2476|anil aggarwal|manager|sales|05/01/59|5000
2365|barun sengupta|director|personnel|05/11/47|7800
1006|chanchal sanghvi|director|sales|09/03/38|6700
……………………………………………………………………
……………………………………………………………………
5678|sumit chakrobarty|d.g.m|marketing|04/19/43|6000
0110|v.k. agrawal |g.m.|marketing|12/31/40|9000
```

Here, the 2nd field *name* is used to sort all the records with the help of –k option. To identify different fields, the delimiter is | and it is used with –t option.

The –r option can be used to sort in the descending order as below –

```
$ sort -t "|" -rk 2 emp.lst
0110|v.k. agrawal |g.m.|marketing|12/31/40|9000
5678|sumit chakrobarty|d.g.m|marketing|04/19/43|6000
```

```
……………………………………………………………………………
2233|a.k. shukla|g.m.|sales|12/12/52|6000
```

- **Sorting on Secondary Key :** One can give more than on key for sorting. That is, one can provide secondary key to sort. If the primary key is the 3rd field (designation), and the secondary key is 2nd field (name), then the command should be given as –

```
$ sort -t "|" -k 3,3 -k 2,2 emp.lst
5423|n.k. gupta|chairman|admin|08/30/56|5400
3212|shyam saksena|d.g.m|accounts|12/12/55|6000
5678|sumit chakrobarty|d.g.m|marketing|04/19/43|6000
2365|barun sengupta|director|personnel|05/11/47|7800
1006|chanchal sanghvi|director|sales|09/03/38|6700
9876|jai sharma|director|production|03/12/50|7000
6521|lalit chowdury|director|marketing|09/26/45|8200
4290|jayant Chodhury|executive|production|09/07/50|6000
3564|sudhir Agarwal|executive|personnel|07/06/47|8000
2233|a.k. shukla|g.m.|sales|12/12/52|6000
2345|j.b. saxena|g.m.|marketing|03/12/45|8000
6213|karuna ganguly|g.m.|accounts|06/05/62|6300
0110|v.k. agrawal |g.m.|marketing|12/31/40|9000
2476|anil aggarwal|manager|sales|05/01/59|5000
1265|s.n. dasgupta|manager|sales|09/12/63|5600
```

Observe the output carefully. We need to sort 3rd field as a primary key (3,3 stands for both starting and ending fields are 3), and then 2nd field as secondary key. The 3rd field *designation* with value `chairman` appears as first record. There are two records with *designation* value as `d.g.m`. Out of these two records, which one has to appear first? Answer is – to be sorted based on 2nd field, *name*. Hence, `shyam saksena` is printed first and then `sumit chakrobarty` is printed. The same logic continues for all the records. Wherever the *designation* field is same, the records are sorted by *name.*

- **Sorting on Columns:** A character position with in a field can be used for sorting the records. For example, assume we would like to sort the file *emp.lst* based on year of birth. Now, year of birth is 7th and 8th column (character) in the 5th field. Hence, the command should be given as –

```
$ sort -t "|" -k 5.7,5.8 emp.lst
1006|chanchal sanghvi|director|sales|09/03/38|6700
0110|v.k. agrawal |g.m.|marketing|12/31/40|9000
5678|sumit chakrobarty|d.g.m|marketing|04/19/43|6000
………………………………………………………………………………
1265|s.n. dasgupta|manager|sales|09/12/63|5600
```

- **Numeric Sort (-n):** Assume that there is a file *numfile* containing only the numbers from 1 to 12. Now if we apply ***sort*** command on this file, the result will be strange –

```
$ sort numfile
1
10
11
12
2
3
………
9
```

This is because, ***sort*** is treating them as strings and as per ASCII collating sequence, 1 appears before 2. To avoid such behavior, *-n* (numeric) option has to be used as below –

```
$ sort –n numfile
1
2
3
………
10
11
12
```

- **Removing Repeated Lines (-u):** The –u (unique) option is used to remove repeated lines from a file. Following example first cuts the designation field from all the records of *emp.lst* and then sorted to find unique designations in the file.

```
$ cut -d "|" -f3 emp.lst | sort -u
chairman
d.g.m
director
executive
g.m.
manager
```

3.8 uniq: Locate Repeated and Non-repeated Lines

Whenever two files are concatenated or merges, there is a chance of duplicate entries. The ***sort –u*** option removes such duplicates. But, there is a special UNIX command for this purpose viz. ***uniq.*** The command requires the input file to be sorted. Hence, it is a practice to ***sort*** the file first and then give its output to ***uniq*** as input through | symbol.

Various options of ***uniq*** command are discussed here –

- **Selecting the Non-repeated Lines (-u):** To determine the uniquely appearing record in the file, ***-u*** option is used. For example, to check a unique designation in emp.lst, cut out the 3rd field (designation), then sort it and then pipe it to ***uniq*** as below –

```
$cut -d "|" -f3 emp.lst |sort | uniq -u
chairman
```

Here, output of *cut* command (3rd field – designation) is given as input to *sort* command. Again, output of *sort* (designation in sorted order) is given as input to *uniq* command. The –u option of *uniq* selects a non-repeated entry out of all designations, which is *chairman* in this example.

- **Selecting the duplicate Lines (-d):** The –d option of *uniq* command is used to display one copy of the repeated/duplicate lines in the file. For example,

```
$ cut -d "|" -f3 emp.lst |sort | uniq -d
d.g.m
director
executive
g.m.
manager
```

- **Counting Frequency of Occurrence(-c):** When there are repeated lines, we may require to know how many times each line is repeated. This is done with –c option as below -

```
$ cut -d "|" -f3 emp.lst |sort | uniq -c
 1 chairman
 2 d.g.m
 4 director
 2 executive
 4 g.m.
 2 manager
```

Here, *chairman* appeared once, *d.g.m.* appeared twice and so on.

3.9 tr: Translating Characters

The ***tr*** command is used to manipulate individual characters in a line. It translates characters using one or two compact expressions. The syntax is –

tr *options expr1 expr2 stdinput*

Note that ***tr*** takes input only from standard input, it doesn't take a file name as argument. By default, it translates each character in *expr1* to its mapped counterpart in *expr2*. The first character in *expr1* is replaced by first character in *expr2* and so on.

Consider an example where we try to replace the delimiter | symbol in *emp.lst* with another character ~ as below –

```
$ tr ‘|’ ‘~’ < emp.lst
2233~a.k. shukla~g.m.~sales~12/12/52~6000
9876~jai sharma~director~production~03/12/50~7000
5678~sumit chakrobarty~d.g.m~marketing~04/19/43~6000
```

```
2365~barun sengupta~director~personnel~05/11/47~7800
……………………………………………
```

Note that, < is a symbol for input redirection. So, here the content of the file *emp.lst* is redirected to standard input and the ***tr*** command receives it process further.

In a single command, one can change more than one character. For example, following command replaces | by ~ and / (used in date of birth field) by –.

```
$ tr ‘|/’ ‘~-’ < emp.lst
2233~a.k. shukla~g.m.~sales~12-12-52~6000
9876~jai sharma~director~production~03-12-50~7000
5678~sumit chakrobarty~d.g.m~marketing~04-19-43~6000
……………………………………………………
```

The ***tr*** command can be used to change the case of the text as below –

```
$ cat emp.lst | tr ‘[a-z]’ ‘[A-Z]’
2233|A.K. SHUKLA|G.M.|SALES|12/12/52|6000
9876|JAI SHARMA|DIRECTOR|PRODUCTION|03/12/50|7000
5678|SUMIT CHAKROBARTY|D.G.M|MARKETING|04/19/43|6000
…………………………………………………………
```

3.9.1 tr Options

Some of the options of ***tr*** command are discussed here –

- **Deleting Characters (-d):** This option is used for deleting some characters from the file. For example, if we would like to delete the separator ‘/’ situated in the date of birth field of *emp.lst* , we can write as below –

  ```
  $ tr –d ‘/’ < emp.lst
  2233|a.k. shukla|g.m.|sales|121252|6000
  9876|jai sharma|director|production|031250|7000
  5678|sumit chakrobarty|d.g.m|marketing|041943|6000
  …………………………………………………
  ```

- **Compressing multiple Consecutive Characters (-s):** Some of the files may have additional white spaces, and we may want to remove them. In such situations, the –s option of *tr* command is useful. As an illustration, create a file as below –

  ```
  $ cat > test
  Hello                    how            are    you?
  I am              fine
  (Ctrl + c)
  ```

  Now, try to remove additional spaces in between as below –

  ```
  $ tr –s ‘ ’ < test
  Hello how are you?
  I am fine
  ```

- **Complementing Values of Expression (-c):** The –c option complements the set of characters in the expression. Thus, to delete all characters except the | and /,

- **Using ASCII Octal Values and Escape Sequences:** The ***tr*** command can use octal values and escape sequences to represent characters. This facility is useful when non-printable characters are there in the file. The following example is used to display each field in a separate line by replacing | symbol by new-line character.

```
$ tr '|' '\n' < emp.lst |head -n 10
2233
a.k. shukla
g.m.
sales
12/12/52
6000
9876
jai sharma
director
production
```

Observe that, we have used *head* command to display only 10 lines.

3.10 FILTERS USING REGULAR EXPRESSIONS

In real time scenario, we often need to search a file to check whether a required pattern exists or not. And we may require to replace certain text with some other text. In UNIX two important commands ***grep*** and ***sed*** takes care of most of the search requirements. To work with these commands for efficient searching, we need to know regular expressions as well. Here we will discuss these concepts in detail.

3.10.1 grep: Searching for a Pattern

The ***grep*** command scans its input for a pattern and displays lines contain the pattern, the line numbers or filenames where the pattern occurs. The syntax is –

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

When filename is not specified, the ***grep*** searches for pattern in standard input. Following example is to search the text *sales* in the file *emp.lst.*

```
$ grep "sales" emp.lst
2233|a.k. shukla|g.m.|sales|12/12/52|6000
1006|chanchal sanghvi|director|sales|09/03/38|6700
1265|s.n. dasgupta|manager|sales|09/12/63|5600
2476|anil aggarwal|manager|sales|05/01/59|5000
```

When the specified pattern is not present in the file, the ***grep*** command displays nothing as shown below –

```
$ grep president emp.lst   #quotes not necessary always
$                          #president not found
```

NOTE that, the pattern need not be put inside double quote always. When the pattern contains more than one word (with white space in-between), it needs to be put inside single/double quotes. When special characters in the patter require command substitution or variable evaluation to be performed, then double quote must be used.

Various options for ***grep*** command are given in Table 3.3. They are discussed with suitable examples below.

Table 3.3 Options for *grep* command

Option	**Description**
-i	Ignores case for matching
-v	Doesn't display lines matching expression
-n	Displays line numbers along with lines
-c	Displays count of number of occurrences
-l	Displays list of filenames only
-e *exp*	Specifies expression with this option. Can use multiple times. Also used for matching expression beginning with a hyphen
-x	Matches pattern with entire line (doesn't match embedded patterns)
-f *file*	Takes patterns from *file,* one per line
-E	Treats pattern as an extended regular expression (ERE)
-F	Matches multiple fixed strings (in **fgrep –** style)

- **Ignoring Case (-i):** When we are searching for a pattern, but not sure about the case, -i option is used. It ignores the case of the text and displays the result. For example,

```
$ grep -i 'agarwal' emp.lst
3564|sudhir Agarwal|executive|personnel|07/06/47|8000
```

- **Deleting Lines (-v):** The –v (inverse) option selects all lines except those containing the pattern. The following example selects all lines in the file *emp.lst* except for those containing the term *director*.

```
$ grep -v 'director' emp.lst
2233|a.k. shukla|g.m.|sales|12/12/52|6000
5678|sumit chakrobarty|d.g.m|marketing|04/19/43|6000
5423|n.k. gupta|chairman|admin|08/30/56|5400
6213|karuna ganguly|g.m.|accounts|06/05/62|6300
1265|s.n. dasgupta|manager|sales|09/12/63|5600
4290|jayant Chodhury|executive|production|09/07/50|6000
2476|anil aggarwal|manager|sales|05/01/59|5000
3212|shyam saksena|d.g.m|accounts|12/12/55|6000
3564|sudhir Agarwal|executive|personnel|07/06/47|8000
2345|j.b. saxena|g.m.|marketing|03/12/45|8000
0110|v.k. agrawal |g.m.|marketing|12/31/40|9000
```

- **Displaying Line Numbers (-n):** This option displays the line numbers containing the pattern along with the actual lines. For example,

```
$ grep -n 'marketing' emp.lst
3:5678|sumit chakrobarty|d.g.m|marketing|04/19/43|6000
11:6521|lalit chowdury|director|marketing|09/26/45|8200
14:2345|j.b. saxena|g.m.|marketing|03/12/45|8000
15:0110|v.k. agrawal |g.m.|marketing|12/31/40|9000
```

- **Counting Lines Containing Pattern (-c):** A pattern may be present in a file multiple times. If we would like to know how many times it has appeared, -c option can be used. The following example shows how many times the pattern *director* has appeared in the file *emp.lst*.

```
$ grep -c 'director' emp.lst
4
```

- **Displaying Filenames (-l):** The –l (el) option is used to display the names of files containing the pattern. Assume there are there are two more files *test.lst* and *testfile.lst* along with *emp.lst*. Now, let us check in which file(s) the pattern *manager* is present.

```
$grep –l ‘manager’ *.lst
emp.lst
test.lst
```

- **Matching Multiple Patterns (-e):** When we would like to search for multiple patterns in a file, we can use –e option. For example,

```
$ grep -e "Agarwal" -e "aggarwal" -e "agrawal" emp.lst
2476|anil aggarwal|manager|sales|05/01/59|5000
3564|sudhir Agarwal|executive|personnel|07/06/47|8000
0110|v.k. agrawal |g.m.|marketing|12/31/40|9000
```

- **Taking Patterns from a File (-f):** If various patterns are stored in a file each in different line, then –f option can be used by giving that filename as one of the arguments. For example, assume there is a file *pattern.lst* as –

```
$cat >pattern.lst
manager
executive
```

Then, give the command as –

```
$grep –f pattern.lst emp.lst
1265|s.n. dasgupta|manager|sales|09/12/63|5600
4290|jayant Chodhury|executive|production|09/07/50|6000
2476|anil aggarwal|manager|sales|05/01/59|5000
3564|sudhir Agarwal|executive|personnel|07/06/47|8000
```

3.11 BASIC REGULAR EXPRESSIONS (BRE)

A regular expression (regex) is defined as a pattern that defines a class of strings. Given a string, we can then test if the string belongs to this class of patterns. Regular expressions are used by many of the UNIX utilities like *grep*, *sed*, *awk*, *vi* etc. A regular expression is a set of characters that specify a pattern. Regular expressions are used when we want to search for specify lines of text containing a particular pattern. Regular expressions search for patterns on a single line, and not for patterns that start on one line and end on another.

The regular expression uses meta-character set as given in Table 3.4.

Table 3.4 Character subset for BRE

Symbol/ Expression	**Matches**
`*`	Zero or more occurrences of the previous character
`g*`	Nothing or g, gg, ggg etc
`.`	A single character
`.*`	Nothing or any number of characters
`[pqr]`	A single character *p, q* or r
`[c1-c2]`	A single character within the ASCII range represented by c1 and c2
`[1-3]`	A digit between 1 and 3
`[^pqr]`	A single character which is not *p, q* or *r*
`[^a-zA-Z]`	A non-alphabetic character
`^pat`	Pattern *pat* at the beginning of line
`pat$`	Pattern *pat* at end of line
`^pat$`	*pat* as only word in line
`^$`	Lines containing nothing

Regular expressions may belong to one of the two categories – *basic* and *extended.* The ***grep*** command supports basic regular expressions (BRE) by default and extended regular expressions (ERE) with –E option. The ***sed*** command supports only BRE set. Initially, we will discuss BRE.

3.11.1 The Character Class

A regular expression lets to specify a group of characters enclosed within a pair of rectangular brackets [ ]. The match is performed for a single character in the group. For example, the expression `[ra]` matches either `r` or a.

In the previous section, we have seen that ***grep*** with –e option is used to compare multiple patterns. Now, let us write the regular expression for searching different spellings of *agarwal* in *emp.lst*.

```
$ grep "[aA]g[ar][ar]wal" emp.lst
3564|sudhir Agarwal|executive|personnel|07/06/47|8000
0110|v.k. agrawal |g.m.|marketing|12/31/40|9000
```

In the above example, the character class `[aA]` matches the letter *a* in both lowercase and uppercase. The model `[ar][ar]` matches any of the four patterns –

```
          aa    ar    ra    rr
```

The pattern [a-zA-Z0-9] matches a single alphanumeric character. While giving range of characters, make sure that the character on the left of the hyphen has lower ASCII value than the character at right.

Regular expressions use the ^ symbol to negate the character class. When the character class begins with this character, all characters other than the ones grouped in the class are matched. So, [^a-zA-Z] matches a single non-alphabetic character string.

3.11.2 The *

The `*` refers to the *immediately preceding* character. It matches *zero or more occurrences of the previous character*. Hence, the pattern `g*` matches *null string* or following strings –

```
     g, gg, ggg, gggg ………
```

As the `*` can match even a null string, if you want to search a string beginning with *g*, do not give pattern as `g*` , instead give as `gg*.`

Now check the following example, where all three types of spellings of *agarwal* can be searched.

```
$ grep "[aA]gg*[ar][ar]wal" emp.lst
2476|anil aggarwal|manager|sales|05/01/59|5000
3564|sudhir Agarwal|executive|personnel|07/06/47|8000
0110|v.k. agrawal |g.m.|marketing|12/31/40|9000
```

3.11.3 The Dot (.)

The dot (`.`) matches a single character. For example, the pattern 2… matches a four-character pattern beginning with a 2. The combination of `*` and dot (`.*`) constitutes a very useful regular expression. It signifies any number of characters or none. For example, when you are not sure about the initial of *saxena,* you can give the expression as -

```
$ grep ".*saxena" emp.lst
2345|j.b. saxena|g.m.|marketing|03/12/45|8000
```

3.11.4 Specifying Pattern Locations (^ and $)

When we need to search for a pattern either at the beginning or at the end of a line, we can use ^ and $ respectively. For example, following command searches all the employees whose employee ID starts with 2.

```
$ grep "^2" emp.lst
2233|a.k. shukla|g.m.|sales|12/12/52|6000
2365|barun sengupta|director|personnel|05/11/47|7800
2476|anil aggarwal|manager|sales|05/01/59|5000
2345|j.b. saxena|g.m.|marketing|03/12/45|8000
```

Similarly, if you would like to search all the employees whose salary is between 7000 and 7999, use the following statement –

```
$grep “7…$” emp.lst
9876|jai sharma|director|production|03/12/50|7000
2365|barun sengupta|director|personnel|05/11/47|7800
```

Here, three dots indicate three positions after 7.

To select only those lines where employee ID do not begin with 2, use the command as below –

```
$ grep “^[^2]” emp.lst
9876|jai sharma|director|production|03/12/50|7000
5678|sumit chakrobarty|d.g.m|marketing|04/19/43|6000
5423|n.k. gupta|chairman|admin|08/30/56|5400
1006|chanchal sanghvi|director|sales|09/03/38|6700
6213|karuna ganguly|g.m.|accounts|06/05/62|6300
1265|s.n. dasgupta|manager|sales|09/12/63|5600
4290|jayant Chodhury|executive|production|09/07/50|6000
6521|lalit chowdury|director|marketing|09/26/45|8200
3212|shyam saksena|d.g.m|accounts|12/12/55|6000
3564|sudhir Agarwal|executive|personnel|07/06/47|8000
0110|v.k. agrawal |g.m.|marketing|12/31/40|9000
```

Observe the regular expression “^[^2]” carefully. Here, [^2] indicates all characters except 2 (as per Table 3.4). Then the ^ in the beginning indicates all records starting with those numbers (except 2). Similarly, the following command extracts all lines where employee ID does not start with any of 2, 3, 5, and 9.

```
$grep “^[^2359]” emp.lst
1006|chanchal sanghvi|director|sales|09/03/38|6700
6213|karuna ganguly|g.m.|accounts|06/05/62|6300
1265|s.n. dasgupta|manager|sales|09/12/63|5600
4290|jayant Chodhury|executive|production|09/07/50|6000
6521|lalit chowdury|director|marketing|09/26/45|8200
0110|v.k. agrawal |g.m.|marketing|12/31/40|9000
```

We will see one more use-case of ^ symbol. As we know, there is no command in UNIX to list only directories. So, if we want to list only directories, we can use *grep* as below –

```
$ ls –l | grep “^d”
drwxrwxr-x 2 chetana chetana 4096 Feb  6  2007 myDir
```

NOTE:

- When the ^ symbol is placed at the beginning of character class (Ex. [^a-z]) it negates every character of the class.
- If ^ symbol is placed outside the character class and at the beginning of the expression (Ex. ^2…), the pattern is matched at the beginning of the line.

- If ^ symbol appears at any other position (Ex. a^bc), then it is treated as a part of pattern itself and matched with itself.

3.12 EXTENDED REGULAR EXPRESSIONS (ERE)

Extended regular expressions make it possible to match dissimilar patterns with a single expression. It uses some additional characters as shown in Table 3.5. In some of the versions of UNIX, the syntax of ERE works with ***grep*** command with –E option. If this is not working, then ***egrep*** command must be used.

Table 3.5 Characters used in Extended Regular Expression

Expression	Description
`ch+`	Matches one or more occurrences of character *ch*
`ch?`	Matches zero or one occurrence of character *ch*
`exp1 | exp2`	Matches *exp1* or *exp2*
`(x1 | x2)x3`	Matches x1x3 or x2x3

3.12.1 The + and ?

The ERE set includes two special characters + and ?, whose meaning is as below –

- The + matches one or more occurrences of the previous character
- The ? matches zero or one occurrence of the previous character.

Thus, the expression *b+* matches any of *b, bb, bbb etc*. And *b?* matches either single instance of *b* or nothing.

Consider the following example –

```
$ egrep "[aA]gg?arwal" emp.lst
2476|anil aggarwal|manager|sales|05/01/59|5000
3564|sudhir Agarwal|executive|personnel|07/06/47|8000
```

3.12.2 Matching Multiple Patterns - |, (and)

The pipe symbol can be used as a delimiter for multiple patterns. For example,

```
$ egrep 'sengupta|dasgupta' emp.lst
2365|barun sengupta|director|personnel|05/11/47|7800
1265|s.n. dasgupta|manager|sales|09/12/63|5600
```

UNIX provides even better alternative using parentheses as shown below –

```
$ egrep '(sen|das)gupta' emp.lst
2365|barun sengupta|director|personnel|05/11/47|7800
1265|s.n. dasgupta|manager|sales|09/12/63|5600
```

The combination of BRE and ERE is a very powerful regular expression.

3.13 sed: THE STREAM EDITOR

The ***sed*** command is a multipurpose tool which combines the work of several filters. It performs non-interactive operations on a data stream. It allows selecting lines and running instructions on them.

An instruction combines an ***address*** for selecting lines, with an ***action*** to be taken on them. The ***sed*** command uses such instructions. The syntax is –

sed *options 'address action' file(s)*

The *address* and *action* are enclosed within single quotes. Addressing in ***sed*** is done in two ways –

- **Line Addressing:** Here, *address* specifies either one line number to select a single line or a set of two numbers to select a group of contiguous lines.
- **Context Addressing:** Here address is specified as a pattern enclosed within a pair of slash (/) (like `/From:/`): It uses one or two patterns for line selection.

The action component is drawn from a set of internal commands of ***sed*** as shown in Table 3.6. The action can be a simple display or an editing function like insertion, deletion, substitution etc. These actions are referred as ***commands.*** The *sed* processes several instructions in a sequential manner. Each instruction operates on the output of the previous instruction.

Table 3.6 Internal Commands of ***sed*** command

Command	Description
`i, a, c`	Inserts, appends and changes text
`d`	Deletes line(s)
`10q`	Quits after reading the first 10 lines
`P`	Prints line(s) on standard output
`3,$p`	Prints lines 3 to end (-n option required)
`$!p`	Prints all lines except last line (-n option required)
`/begin/, /end/p`	Prints lines enclosed between *begin* and *end* (-n option required)
`q`	Quits after reading up to addressed line
`r` *`flname`*	Places contents of file *flname* after line
`w` *`flname`*	Writes addressed lines to file *flname*
`=`	Prints line number addressed.
`s/s1/s2/`	Replaces first occurrence of expression s1 in all lines with expression s2
`10,20s/-/:/`	Replaces first occurrence of – in lines 10 to 20 with a :
`s/s1/s2/g`	Replaces all occurrences of expression s1 in all lines with expression s2
`s/-/:/g`	Replaces all occurrences of – in all lines with a :

3.13.1 Line Addressing

In line addressing, the instruction *3q* can be broken into the address 3 and the action q (quit). So, to display only first 3 lines, (similar to *head –n 3*) use the following statement –

```
$ sed '3q' emp.lst
2233|a.k. shukla|g.m.|sales|12/12/52|6000
9876|jai sharma|director|production|03/12/50|7000
5678|sumit chakrobarty|d.g.m|marketing|04/19/43|6000
```

In the above example, 3 lines will be displayed and then quits.

Generally, the ***p*** (print) command is used to display lines. But, this command behaves strange – it prints selected lines as well as *all* lines. Hence, the selected lines will appear twice. To suppress this feature of *p*, the –n option has to be used. The following example selects the lines 5 through 7.

```
$ sed  -n '5,7p' emp.lst
5423|n.k. gupta|chairman|admin|08/30/56|5400
1006|chanchal sanghvi|director|sales|09/03/38|6700
6213|karuna ganguly|g.m.|accounts|06/05/62|6300
```

The $ symbol can be used to print only the last line as below –

```
$ sed -n '$p' emp.lst
0110|v.k. agrawal |g.m.|marketing|12/31/40|9000
```

The ***sed*** command can be used to select multiple groups of lines. In that case, each address has to be given in a different line, but enclosed within a single pair of quotes as shown below –

```
$ sed -n '1,2p
> 7,9p
> $p' emp.lst
2233|a.k. shukla|g.m.|sales|12/12/52|6000
9876|jai sharma|director|production|03/12/50|7000
6213|karuna ganguly|g.m.|accounts|06/05/62|6300
1265|s.n. dasgupta|manager|sales|09/12/63|5600
4290|jayant Chodhury|executive|production|09/07/50|6000
0110|v.k. agrawal |g.m.|marketing|12/31/40|9000
```

The ***sed*** command uses `!` (exclamatory mark) as a negation operator. Assume, we would like to select first 2 lines of the file. Note that, selecting first two lines means – not selecting 3rd line to end. So, the command can be used as below –

```
$ sed -n '3,$!p' emp.lst
2233|a.k. shukla|g.m.|sales|12/12/52|6000
9876|jai sharma|director|production|03/12/50|7000
```

Here, `!` is for ***p*** indicating not to print lines from 3 to end.

Using Multiple Instructions (-e and –f) : In the previous section, we have seen that when multiple groups of lines have to be selected, the pattern should be given in different lines with a line-break in-between. To avoid that, ***sed*** uses –e option. This option allows to enter as many instructions as you wish, in a single line, where each instruction is preceded by the option –e. For example, the following command selects multiple lines (1 to 2, 7 to 9 and last line) –

```
$ sed -n -e '1,2p' -e '7,9p' -e '$p' emp.lst
2233|a.k. shukla|g.m.|sales|12/12/52|6000
9876|jai sharma|director|production|03/12/50|7000
6213|karuna ganguly|g.m.|accounts|06/05/62|6300
1265|s.n. dasgupta|manager|sales|09/12/63|5600
4290|jayant Chodhury|executive|production|09/07/50|6000
0110|v.k. agrawal |g.m.|marketing|12/31/40|9000
```

When we have too many instructions to use or when we have a set of a common instructions that are executed often, better to store them in a file. And, then use –f option with ***sed*** command to read from that file and to apply the instructions on input file. Consider the example given below. Here, we have created a file *instr.dat* containing required instructions. Then use the ***sed*** command.

```
$ cat >instr.dat
     1,2p
     7,9p
     $p

$ sed –n –f instr.dat emp.lst
2233|a.k. shukla|g.m.|sales|12/12/52|6000
9876|jai sharma|director|production|03/12/50|7000
6213|karuna ganguly|g.m.|accounts|06/05/62|6300
1265|s.n. dasgupta|manager|sales|09/12/63|5600
4290|jayant Chodhury|executive|production|09/07/50|6000
0110|v.k. agrawal |g.m.|marketing|12/31/40|9000
```

NOTE:

- The –f option can be used to read instructions from multiple files. For example, if you have two files *instr1.dat* and *instr2.dat* containing instructions, then use –

  ```
  $sed –n –f instr1.dat –f instr2.dat emp.lst
  ```

- Both -e and –f options can be used with a single ***sed*** command. For example,

  ```
  $sed –n –e ‘3,4p’ –f instr.dat emp.lst
  ```

3.13.2 Context Addressing

Context addressing allows to specify one or two patterns to locate the lines. The patterns must be bounded by a / on both the sides. When a single pattern is specified, all lines containing the pattern are selected. The following example is for selecting all the lines containing the pattern *director*.

```
$ sed -n '/director/p' emp.lst
9876|jai sharma|director|production|03/12/50|7000
2365|barun sengupta|director|personnel|05/11/47|7800
1006|chanchal sanghvi|director|sales|09/03/38|6700
6521|lalit chowdury|director|marketing|09/26/45|8200
```

One can give a comma-separated list of context addresses to select a group of lines. For example, to select all the lines between *dasgupta* and *saxena* use the following statement –

```
$ sed -n '/dasgupta/,/saksena/p' emp.lst
1265|s.n. dasgupta|manager|sales|09/12/63|5600
4290|jayant Chodhury|executive|production|09/07/50|6000
2476|anil aggarwal|manager|sales|05/01/59|5000
6521|lalit chowdury|director|marketing|09/26/45|8200
3212|shyam saksena|d.g.m|accounts|12/12/55|6000
```

One can mix line addressing and context addressing. If we want to select all lines from 1st line till *dasgupta*, use the command as below –

```
$ sed -n '1,/dasgupta/p' emp.lst
2233|a.k. shukla|g.m.|sales|12/12/52|6000
9876|jai sharma|director|production|03/12/50|7000
5678|sumit chakrobarty|d.g.m|marketing|04/19/43|6000
2365|barun sengupta|director|personnel|05/11/47|7800
5423|n.k. gupta|chairman|admin|08/30/56|5400
1006|chanchal sanghvi|director|sales|09/03/38|6700
6213|karuna ganguly|g.m.|accounts|06/05/62|6300
1265|s.n. dasgupta|manager|sales|09/12/63|5600
```

Regular expressions can be used as a part of context address. For example, the following command selects different spellings of *agarwal.*

```
$ sed -n '/[aA]gg*[ar][ar]wal/p' emp.lst
2476|anil aggarwal|manager|sales|05/01/59|5000
3564|sudhir Agarwal|executive|personnel|07/06/47|8000
0110|v.k. agrawal |g.m.|marketing|12/31/40|9000
```

One more example of ***sed*** command including regular expression is given below. It selects all lines containing *saksena, saxena,* and *gupta*. Note that, here also two different patterns should be given on different lines.

```
$ sed -n '/sa[kx]s*ena/p
> /gupta/p' emp.lst
2365|barun sengupta|director|personnel|05/11/47|7800
5423|n.k. gupta|chairman|admin|08/30/56|5400
1265|s.n. dasgupta|manager|sales|09/12/63|5600
3212|shyam saksena|d.g.m|accounts|12/12/55|6000
2345|j.b. saxena|g.m.|marketing|03/12/45|8000
```

The characters ^ and $ also can be used as a part of regular expression with ***sed*** command. Following example shows the people born in 1950. Note that, the five dots after 50 in the expressions indicate 5 characters (a delimiter | and 4 characters indicating salary) present before the end of line ($).

```
$ sed -n '/50.....$/p' emp.lst
9876|jai sharma|director|production|03/12/50|7000
4290|jayant Chodhury|executive|production|09/07/50|6000
```

3.13.3 Writing Selected Lines to a File (w)

The lines selected through line addressing or context addressing can be stored in a separate file using ***w*** (write) command. Following example selects all lines containing *director* from the file *emp.lst* and then writes them into a new file *dlist*.

```
$ sed -n '/director/w dlist' emp.lst
$ cat dlist
9876|jai sharma|director|production|03/12/50|7000
2365|barun sengupta|director|personnel|05/11/47|7800
1006|chanchal sanghvi|director|sales|09/03/38|6700
6521|lalit chowdury|director|marketing|09/26/45|8200
```

Lines selected based on different patterns can be stored in different files as shown below. Here, the lines containing *director, manager* and *executive* will be stored in files *dlist*, *mlist* and *elist* respectively.

```
$ sed -n '/director/w dlist
> /manager/w mlist
> /executive/w elist' emp.lst
```

The similar result can be achieved in line addressing as well. The lines from 1 to 7 will be written into *e1.lst* and lines from 8 to 15 are written into *e2.lst* using the following command–

```
$ sed -n '1,7w e1.lst
> 8,15w e2.lst' emp.lst
$                            #use cat command to see e1, e2
```

3.13.4 Text Editing

The ***sed*** command provides some text editing commands as a part of its action component. One can use ***i*** (insert), ***a*** (append), ***c*** (change) and ***d*** (delete) for doing appropriate action on the file. Since ***sed*** is a stream editor, the effect of these commands is **on every line of the input file by default.** If we want the command to be applied on a specific line, then the line number (termed as address) should be specified. The input file will be usually opened for reading. But, the actions like insert, append etc. are writing jobs. It is obvious that a file cannot be opened for reading as well as writing at a time. Hence, the output of these actions must be redirected to a temporary file first. The contents of temporary file must be moved to the input file to modify it using ***mv*** command.

For understanding ***i, a, d*** and ***c*** commands, let us use a file *test.lst* as below –

```
$cat test.lst
Manager
Director
Executive
```

Insertion: Use the command ***i*** to insert any number of lines into a file at a required position. We will consider different examples to understand the working of ***i*** command.

Ex1: Using ***i*** without any address inserts the given line(s) ***before every line*** of the file. For example,

```
$sed ‘i Engineer’ test.lst
Engineer
Manager
Engineer
Director
Engineer
Executive
```

One can observe that *Engineer* has been included before every line of the file. But, if you check the contents of the file *test.lst*, it will be unmodified –

```
$cat test.lst
Manager
Director
Executive
```

This happens because, the ***sed*** command could able to open the input file *test.lst* only for *read* mode and the *insert* requires it to be opened in *write* mode. Hence, the modified contents has to be redirected to a temporary file and then moved to *test.lst* as shown below–

```
$sed ‘i Engineer’ test.lst >temp     #redirecting to temp
$mv temp test.lst                    #moving temp to test.lst

$cat test.lst                        #display test.lst
```

```
     Engineer
     Manager
     Engineer
     Director
     Engineer
     Executive
```

But, in reality, we don’t want to insert *Engineer* at every line. We want it to be inserted in a particular position. The following example inserts the string *Engineer* into 2nd position.

Ex2:

```
     $sed ‘2i Engineer’ test.lst >temp    #redirecting to temp
     $mv temp test.lst                    #moving temp to test.lst
     $cat test.lst                        #display test.lst
     Manager
     Engineer
     Director
     Executive
```

Ex3: One can insert more than one string at a required position using single command. Following example inserts 3 strings into 2nd position of the file.

```
     $sed ‘2i\                  #Give \ before pressing enter key
     >Software Engineer\        #except for the last line
     >Test Engineer\
     >CEO’ test.lst > temp      #Copy result into temporary file

     $ mv temp test.lst         #Move temp file to test.lst
     $ cat test.lst             #Check the contents of test.lst
          Manager
          Software Engineer
          Test Engineer
          CEO
          Director
          Executive
```

Ex4: Using single command, multiple strings can be inserted at multiple positions as shown below –

```
     $sed ‘2i Engineer               #press enter key
     > 3i CEO’ test.lst >temp        #redirecting to temp
     $mv temp test.lst            #moving temp to test.lst
     $cat test.lst                #display test.lst
     Manager
     Engineer
     Director
     CEO
     Executive
```

Append: The command ***a*** is used to append any number of lines at specified position. We will consider various situations of using ***a.***

Ex1. By default, the command ***a*** appends the given string ***after every line*** of the input file. For example,

```
$sed ‘a Engineer’ test.lst > temp
$ mv temp test.lst
$ cat test.lst
     Manager
     Engineer
     Director
     Engineer
     Executive
     Engineer
```

Ex2. To append the string at required position, use the address (line number) as shown below. Here, the string will be appended after 2nd line of *test.lst*.

```
$sed ‘2a Engineer’ test.lst > temp
$ mv temp test.lst
$ cat test.lst
     Manager
     Director
     Engineer
     Executive
```

Ex3. The append command can be used to add line-spacing between the lines of given file–

```
$sed ‘a\                    #press enter key
> ’ test.lst > temp         #close single-quote without any text
$ mv temp test.lst
$ cat test.lst
     Manager

     Director

     Executive
```

In the above example, we have pressed enter-key after giving the command ***a***. The enter-key here acts a as new-line character and it is treated as a string to be appended. Hence it induces one blank-line between every line of the input file.

Change: Use the command ***c*** to change a particular line by required string. By default, the command ***c*** will change all the lines when address is not given. For example,

Ex1.

```
$sed 'c Engineer' test.lst > temp
$ mv temp test.lst
$ cat test.lst
     Engineer
     Engineer
     Engineer
```

Here, we can observe that all the lines of the file *test.lst* got changed to *Engineer*.

Ex2. Change only the required line by specifying the address as below –

```
$sed '3c Deputy Managaer' test.lst > temp
$ mv temp test.lst
$ cat test.lst
     Manager
     Director
     Deputy Manager          #Note the change here
```

Delete: The delete (***d***) command can be used to delete any line by specifying either the line number (address) or the pattern. Various situations are explained with examples here.

Ex1. When ***d*** is used without any address or pattern, it deletes all the lines in the file as shown below –

```
$sed 'd' test.lst > temp      #redirect output to temp
$cat temp                     #display contents of temp
$                   #nothing is displayed as temp is empty
```

Ex2. To delete a particular line, give the line number with ***d.***

```
$sed '2d' test.lst > temp     #delete the 2nd line in test.lst
$mv temp test.lst             #move temp to test.lst
$cat test.lst                 #display test.lst
     Manager                  #2nd line Director is deleted
     Executive
```

Ex3. One can use context addressing and specify the required pattern to be deleted. In the following example, all the lines containing the pattern *Director* will be deleted and the output is stored in the file *olist.lst.*

```
$sed '/director/d' emp.lst > dlist.lst
$cat dlist.lst
2233|a.k. shukla|g.m.|sales|12/12/52|6000
5678|sumit chakrobarty|d.g.m|marketing|04/19/43|6000
5423|n.k. gupta|chairman|admin|08/30/56|5400
6213|karuna ganguly|g.m.|accounts|06/05/62|6300
1265|s.n. dasgupta|manager|sales|09/12/63|5600
4290|jayant Chodhury|executive|production|09/07/50|6000
2476|anil aggarwal|manager|sales|05/01/59|5000
```

```
3212|shyam saksena|d.g.m|accounts|12/12/55|6000
3564|sudhir Agarwal|executive|personnel|07/06/47|8000
2345|j.b. saxena|g.m.|marketing|03/12/45|8000
0110|v.k. agrawal |g.m.|marketing|12/31/40|9000
```

The above result can also be achieved by using the following command –

```
$sed –n '/director/!p' emp.lst > dlist.lst
```

3.13.5 Substitution (s)

The command ***s*** (substitution) allows to replace a pattern in the input file with something else. The syntax is –

[address] **s** */ expr1 / expr2 / flags*

Here, the *expr1* is replaced by *expr2* in all the lines specified by *address*. When *address* is not specified, the substitution is performed for all matching lines in the file. Consider an example –

```
$ sed 's/|/:/' emp.lst
2233:a.k. shukla|g.m.|sales|12/12/52|6000
9876:jai sharma|director|production|03/12/50|7000
5678:sumit chakrobarty|d.g.m|marketing|04/19/43|6000
..........................................................................................
```

Here, our *expr1* is pipe symbol and *expr2* is colon. We are instructing to replace all pipes by colon in the file *emp.lst*. But, when we observe the output, only the first (left-most) occurrence of pipe in every line is replaced by colon. To replace all the pipes in a line, we need to use the flag ***g (global)***. For example,

```
$ sed 's/|/:/g' emp.lst
2233:a.k. shukla:g.m.:sales:12/12/52:6000
9876:jai sharma:director:production:03/12/50:7000
.......................................................................
```

We can choose the number of lines on which the replacement should happen. In the below example, the pipe is replaced by colon only for first two lines –

```
$ sed '1,2s/|/:/g' emp.lst
2233:a.k. shukla:g.m.:sales:12/12/52:6000
9876:jai sharma:director:production:03/12/50:7000
5678|sumit chakrobarty|d.g.m|marketing|04/19/43|6000
2365|barun sengupta|director|personnel|05/11/47|7800
..................................................................................................
```

One can replace a string with another string. In the following example, the string *director* is replaced by *member* only in first 5 lines.

```
$ sed '1,5s/director/member /' emp.lst
2233|a.k. shukla|g.m.|sales|12/12/52|6000
9876|jai sharma|member |production|03/12/50|7000
```

```
5678|sumit chakrobarty|d.g.m|marketing|04/19/43|6000
2365|barun sengupta|member |personnel|05/11/47|7800
5423|n.k. gupta|chairman|admin|08/30/56|5400
1006|chanchal sanghvi|director|sales|09/03/38|6700
……………………………………………………………………………………
```

Regular expressions can be used for patterns while doing substitution. For example, all different spellings like *agarwal, aggarwal* and *agrawal* can all be replaced by one simple string *Agarwal* as shown below –

```
$ sed 's/[Aa]gg*[ar][ar]wal/Agarwal/g' emp.lst
2233|a.k. shukla|g.m.|sales|12/12/52|6000
…………………………………………………………………………
2476|anil Agarwal|manager|sales|05/01/59|5000
……………………………………………………………………………………
3564|sudhir Agarwal|executive|personnel|07/06/47|8000
……………………………………………………………………………………….
0110|v.k. Agarwal |g.m.|marketing|12/31/40|9000
```

The anchoring characters ^ and $ can also be used to indicate beginning and ending of the line in a file. For example, the following statement adds 2 as a prefix to every employee id in the file –

```
$ sed 's/^/2/' emp.lst
22233|a.k. shukla|g.m.|sales|12/12/52|6000
29876|jai sharma|director|production|03/12/50|7000
25678|sumit chakrobarty|d.g.m|marketing|04/19/43|6000
……………………………………………………………………………………
```

The salary of every employee can be suffixed with **.00** using $ symbol as below –

```
$ sed 's/$/.00/' emp.lst
2233|a.k. shukla|g.m.|sales|12/12/52|6000.00
9876|jai sharma|director|production|03/12/50|7000.00
5678|sumit chakrobarty|d.g.m|marketing|04/19/43|6000.00
2365|barun sengupta|director|personnel|05/11/47|7800.00
……………………………………………………………………………………
```

Using a single command, multiple strings can be substituted. For example, we would like to replace *director* by *member*, *executive* by *Execom* and *d.g.m* by *DGM*. Then use the command as –

```
$ sed 's/director/member/g           #press enter key
> s/executive/Execom/g                  #press enter key
> s/d.g.m/DGM/g' emp.lst
……………………………………………………………………………………….
9876|jai sharma|member|production|03/12/50|7000
5678|sumit chakrobarty|DGM|marketing|04/19/43|6000
```

```
2365|barun sengupta|member|personnel|05/11/47|7800
…………………………………………………………………………………………
4290|jayant Chodhury|Execom|production|09/07/50|6000
……………………………………………………………………………………………
```

3.14 BASIC REGULAR EXPRESSIONS REVISTED

We have discussed ***grep*** and ***sed*** command and also the characters used in regular expressions. But, the BRE set contains few more characters. Here we will discuss various characters in regular expressions and three types of expressions as below –

- The *repeated pattern* uses single symbol - & to make the entire source pattern appear at the destination also.
- The *interval regular expression (IRE)* uses the characters { and } with a single or a pair of numbers between them.
- The *tagged regular expression (TRE)* groups the patterns with (and), and represents them at the destination with numbered tags.

In IRE and TRE, the meta-characters needs to be escaped.

3.14.1 The Repeated Pattern (&)

There are some situations where the source pattern also occurs at the destination. The special character & is used to represent it. Consider an example –

To replace *director* by *executive director*, we can use the ***sed*** command as –

```
$sed ‘s/director/executive director/’ emp.lst
```

But, here the destination string *executive director* contains the source string *director* also. Hence, we can used the repeated pattern & in the command as –

```
$ sed 's/director/executive &/' emp.lst
2233|a.k. shukla|g.m.|sales|12/12/52|6000
9876|jai sharma|executive director|production|03/12/50|7000
5678|sumit chakrobarty|d.g.m|marketing|04/19/43|6000
2365|barun sengupta|executive director|personnel|05/11/47|7800
………………………………………………………………………………………
```

The symbol & (known as repeated pattern) here expands to the entire source string.

3.14.2 Interval Regular Expression (IRE)

We have used ^ and $ symbols to match a pattern at the beginning and end of a line in ***sed*** and ***grep*** commands. But how to match a pattern at any position of the line? The answer is – interval regular expression. IRE uses integer(s) to specify the number of characters preceding a pattern for matching any pattern in the line. It takes one of the three forms as below –

- `ch\{m\}` - The meta-character `ch` can occur *m* times
- `ch\{m,n\}` – `ch` can occur between *m* and *n* times

- `ch\{m,\}` – ch can occur at least *m* times

Here, the character *ch* can be a single character, or a dot, or a character class. The values of *m* and *n* should not exceed 255.

For understanding the IRE, let us use the input file *teldir.txt* as given below –

```
$cat teldir.txt
jai sharma 26703289
chanchal singvi 97381356722
anil agarwal 8456012983
shyam saksena 28608110
lalit chowdury 23245194
```

The above file is a telephone directory containing five records, of which two records contain mobile numbers (10 digits) and three records are of land-line numbers (8 digits). Now use the following regular expression with *grep* command to select the records with mobile numbers.

```
$grep ‘[0-9]\{10\}’ teldir.txt
chanchal singvi 97381356722
anil agarwal 8456012983
```

Here, `\{` and `\}` are the flower brackets with escape sequence character `\`. We are instructing to search for the pattern containing digits 0-9 (using character class) and the character would have appeared 10 times.

Consider one more example – listing all the files in the current directory whose *write* (w) bit is set for either group or others.

```
$ ls -l | sed -n '/^.\{5,8\}w/p'
-rw-r--rw- 1 john john  280 Jan 30  2007 caseEx.sh
-rw-rw-r-- 1 john john  104 Feb  3  2007 cmdArg.sh
-rw-rw-rw- 1 john STAFF 741 Jan  3  2007 emp.lst
-rw-r--rw- 1 john john  199 Jan 29  2007 ifEx.sh
-rw-rw-r-- 1 john john  217 Jan 19  2007 logfile
```

Let us understand command step-by-step. Here, the output of *ls –l* is sent as input to *sed* command. In the *sed* command we are using *p* (at the end) to print the selected lines and the option –n is used to avoid *p* from printing all the lines apart from the selected lines. Now, let us concentrate on remaining portion of the regular expression –

```
'/^.\{5,8\}w/p'
```

- / in the beginning and at the end (before p) is used as if we use it in context addressing to enclose the pattern.
- ^ indicates we are searching from the beginning of a line.

- Dot indicates any character
- \{ and \} indicates flower brackets with escape sequence, which is the syntax of IRE.
- 5, 8 indicates there may be 5 to 8 dots.
- w indicates that after 5 to 8 dots, there must be the character *w*.

Now, try to recollect that in the output of *ls –l*, the *w* position for *group* appears after 5 characters and *w* position for *others* appears after 8 characters. Thus, the above regular expression instructs to select the lines in which *w* is present either at 6th position or at 9th position. As we are not bothered about other permissions like *r* and *x,* we are just using dots to skip those characters.

3.14.3 Tagged Regular Expression (TRE)

Tagged regular expressions are most complex regular expressions. It suggests breaking up a line into groups and then extracting one or more of these groups. It requires two regular expressions one each for the source and target pattern. The working of TRE is explained here –

1. Identify the segments of a line that you would like to extract and enclose each segment with a matched pair of escaped parentheses. For example, if you need to extract a number, represent it as \([0-9]*\). A sequence of non-alphabetic characters are represented as \([^a-zA-Z]*\).
2. Every grouped pattern automatically acquires the numeric label *n* indicating nth group from the left.
3. To reproduce a group at the destination, use the tag \n. For example, \1 indicates first group and so on.

Let us consider one example of TRE. Let us use the file *teldir.txt* used in the previous section, which contains 5 records. In this file, records are organized as –

```
Firstname Lastname PhoneNo
```

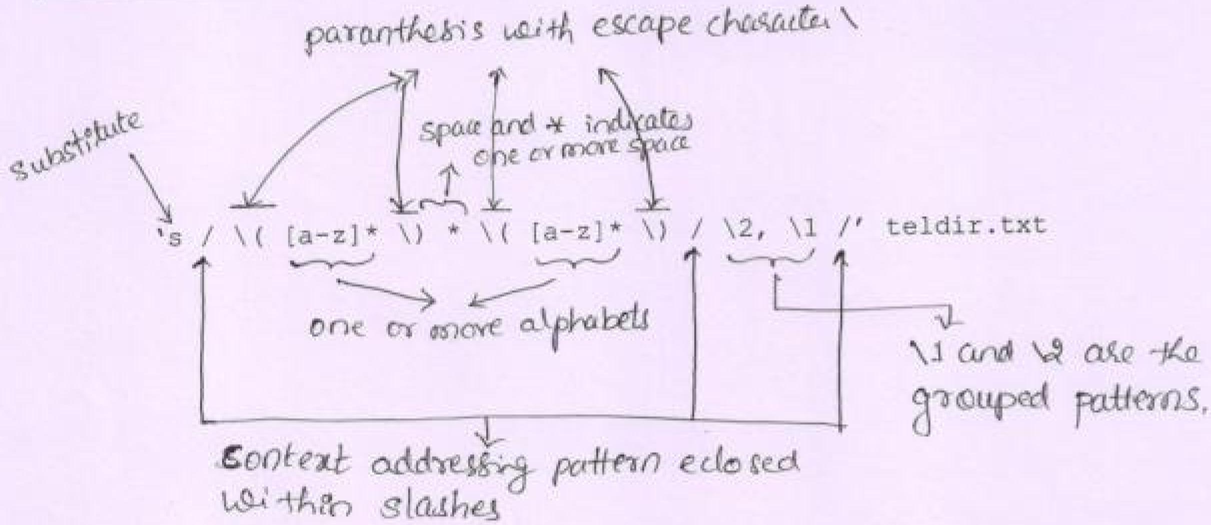
Let us write the regular expression (TRE) which alters the file such that the name of the people in the file will appear as –

```
Lastname, Firstname PhoneNo
```

The command used is as below –

```
$sed ‘s/\([a-z]*\) *\([a-z]*\)/\2, \1/’ teldir.txt
sharma, jai 26703289
singvi, chanchal 97381356722
agarwal, anil 8456012983
saksena, shyam 28608110
chowdury, lalit 23245194
```

Now, let us understand the working of the given command in detail. Let us observe every character (or group of characters) used in the above command.



```
's / \( [a-z]* \) * \( [a-z]* \) / \2, \1 /' teldir.txt
```

paranthesis with escape character \
substitute
space and * indicates one or more space
one or more alphabets
\1 and \2 are the grouped patterns.
Context addressing pattern eclosed within slashes

Thus, \([a-z]*\) is one group of characters (Ex. jai, chanchal etc) representing first name.

<space> * is any no of white spaces between first name and last name.

\(a-z)*\) is second group of characters representing last name (Ex. sharma, singrei etc)

\2, \1 indicates that the output should be like second group, first group.
i.e. last name, first name