Chapter-6 Customization and Filters (8L)

TOPICS 2. Filters

Prepare file for printing (pr), Custom display of file using head and tail, Vertical division of file (cut), Paste files (paste),

Sort file (sort), Finding repetition and non- repetition (uniq), Manipulating characters using tr, Searching pattern using grep,

Brief idea of using Basic Regular Expression (BRE), Extended Regular Expression (ERE), and egrep,

grep –E

TOPICS 2. Filters:

pr command in Linux:

Prepare file for printing (pr):

In Linux/Unix pr command is used to prepare a file for printing by adding suitable footers, headers, and the formatted text.

pr command actually **adds 5 lines of margin both at the top and bottom of the page**. The header part shows the date and time of the last modification of the file with the file name and the page number.

NAME

pr - convert text files for printing

Syntax:

```
pr [options][filename]
```

Working with pr command:

1. To print k number of columns we use -k.

Let's say, we have a file that contains 5 numbers from 1 to 5 with every number in a new line

```
debasis@LAPTOP-H3N6JCNE:~$ cat -n pr.txt
     1  this is my test file
     2  BCA
     3  MCA
     4  CSE
     5  IT
```

1.Now if we want to print this **content in 2 columns** we will use the following command.

```
debasis@LAPTOP-H3N6JCNE:~$ pr -2 pr.txt | more
```

output:

```
2020-12-10 19:29              pr.txt              Page 1
```

this is my test file CSE
BCA IT
MCA

Here pr.txt is the name of the file.

2. To **suppress the headers and footers** the -t option is used

```
debasis@LAPTOP-H3N6JCNE:~$ pr -t pr.txt | more

this is my test file
BCA
MCA
CSE
IT
```

3. To Double the spaces input, reduces clutter -d option is used.

```
debasis@LAPTOP-H3N6JCNE:~$ pr -d pr.txt | more

2020-12-10 19:29                pr.txt                Page 1


this is my test file

BCA

MCA

CSE

IT
```

4. To provide number of lines which helps in debugging the code , -n option is used.

```
debasis@LAPTOP-H3N6JCNE:~$ pr -n pr.txt | more

2020-12-10 19:29                pr.txt                Page 1


     1  this is my test file
     2  BCA
     3  MCA
     4  CSE
     5  IT
```

Custom display of file using head and tail:

Head command in Linux with examples:

It is the complementary of tail command. The head command, as the name implies, print the top N number of data of the given input.

By default, it prints the first 10 lines of the specified files. If more than one file name is provided then data from each file is preceded by its file name.

Syntax:

```
head [OPTION]... [FILE]...
```

Without any option, it displays only the first 10 lines of the file specified.
Example:

```
$ head filename.txt
Andhra Pradesh
Arunachal Pradesh
Assam
Bihar
Chhattisgarh
Goa
Gujarat
Haryana
Himachal Pradesh
Jammu and Kashmir
```

Without any option with head, it displays only the first 10 lines

```
debasis@LAPTOP-H3N6JCNE:~$ ls -l | head
total 80
-rw-r--r-- 1 debasis debasis   254 Nov 30 14:34 '
-rw-r--r-- 1 debasis debasis     0 Nov 26 13:04 0
-rw-r--r-- 1 debasis debasis     2 Nov 13 10:31 30
-rw-r--r-- 1 debasis debasis     0 Nov 13 10:46 =
drwxr-xr-x 1 debasis debasis  4096 Dec 10 12:15 BCA
drwxr-xr-x 1 debasis debasis  4096 Dec 10 10:00 CSE
drwxr-xr-x 1 debasis debasis  4096 Nov 11 16:11 CSE10
drwxr-xr-x 1 debasis debasis  4096 Nov 11 15:27 CSE2
drwxr-xr-x 1 debasis debasis  4096 Dec  2 16:22 CSE20
```

1. -n num: Prints the first 'num' lines instead of first 10 lines. **num** is mandatory to be specified in command otherwise it displays an error.

$ head -n 5 filename.txt

```
Andhra Pradesh
Arunachal Pradesh
Assam
```

```
Bihar
Chhattisgarh
```

```
Print the top FIVE Lines:
debasis@LAPTOP-H3N6JCNE:~$ ls -l | head -5
total 80
-rw-r--r-- 1 debasis debasis   254 Nov 30 14:34 '
-rw-r--r-- 1 debasis debasis     0 Nov 26 13:04 0
-rw-r--r-- 1 debasis debasis     2 Nov 13 10:31 30
-rw-r--r-- 1 debasis debasis     0 Nov 13 10:46 =
```

tail command :

It is the complementary of head command. The tail command, as the name implies, print the last N number of data of the given input.

By default it prints the last 10 lines of the specified files. If more than one file name is provided then data from each file is precedes by its file name.

Syntax:

```
tail [OPTION]... [FILE]...
```

Without any option it display only the last 10 lines of the file specified.
Example:

```
$ tail filename.txt
Odisha
Punjab
Rajasthan
Sikkim
Tamil Nadu
Telangana
Tripura
Uttar Pradesh
Uttarakhand
West Bengal
```

```
Print last TEN Lines(default):

debasis@LAPTOP-H3N6JCNE:~$ ls -l | tail
-rw-r--r-- 1 debasis debasis   603 Nov 14 15:16 temp.sh
-rw-r--r-- 1 debasis debasis   602 Nov 18 20:07 temperature.sh
-rw-r--r-- 1 debasis debasis   296 Nov 13 12:29 test1.sh
-rw-r--r-- 1 debasis debasis   190 Nov 19 19:30 tmp.sh
-rw-r--r-- 1 debasis debasis   102 Dec  9 15:09 wh.sh
-rw-r--r-- 1 debasis debasis   300 Dec  9 15:13 wh_con.sh
-rw-r--r-- 1 debasis debasis   191 Dec  9 15:21 wh_con10.sh
-rw-r--r-- 1 debasis debasis   210 Dec  9 15:16 wh_con2.sh
-rw-r--r-- 1 debasis debasis   164 Dec  9 15:11 whh.sh
-rw-r--r-- 1 debasis debasis   184 Nov 11 14:28 xyz1.txt
```

-n num: Prints the last 'num' lines instead of last 10 lines. **num** is mandatory to be specified in command otherwise it displays an error. This command can also be written as without symbolizing 'n' character but '-' sign is mandatory.

```
$ tail -n 3 filename.txt
```

```
Uttar Pradesh
Uttarakhand
West Bengal
       OR
$ tail -3 filename.txt
Uttar Pradesh
Uttarakhand
West Bengal
```

Print last THREE Lines:

```
debasis@LAPTOP-H3N6JCNE:~$ ls -l | tail -n 3
-rw-r--r-- 1 debasis debasis   210 Dec  9 15:16 wh_con2.sh
-rw-r--r-- 1 debasis debasis   164 Dec  9 15:11 whh.sh
-rw-r--r-- 1 debasis debasis   184 Nov 11 14:28 xyz1.txt
```

Print last THREE Lines:

```
debasis@LAPTOP-H3N6JCNE:~$ ls -l | tail -3
-rw-r--r-- 1 debasis debasis   210 Dec  9 15:16 wh_con2.sh
-rw-r--r-- 1 debasis debasis   164 Dec  9 15:11 whh.sh
-rw-r--r-- 1 debasis debasis   184 Nov 11 14:28 xyz1.txt
```

Applications of head and tail Command

Print line between M and N lines:

For this purpose we use head, tail and pipeline(|) commands.

Command is:

head -M file_name | tail -(M-N+1),

since the first line takes first M lines and tail command cuts (M-N+1) Lines starting from the end.

Q. Let say from filename.txt file we have to print lines between 10 and 20.

M=20

N=10

Tail=M-N+1=20-10+1=11

$ head -n 20 filename.txt | tail –n 11

```
1. Jharkhand
2.
3. Karnataka
4. Kerala
5. Madhya Pradesh
6. Maharashtra
7. Manipur
8. Meghalaya
9. Mizoram
10. Nagaland
11. Odisha
```

```
#Display from 10 to 20 numbers
M=20
N=10
Tail=M-N+1=20-10+1=11

debasis@LAPTOP-H3N6JCNE:~$ alias 10to20='ls -l | head -n 20 | tail -n 11'
debasis@LAPTOP-H3N6JCNE:~$ 10to20

output:
drwxr-xr-x 1 debasis debasis  4096 Dec  2 16:22 CSE20
drwxr-xr-x 1 debasis debasis  4096 Nov 12 11:44 MCA
-rw-r--r-- 1 debasis debasis    10 Nov 13 11:00 a
-rw-r--r-- 1 debasis debasis    76 Nov  2 10:43 a.c
-rwxr-xr-x 1 debasis debasis 16696 Oct 29 12:04 a.out
-rw-r--r-- 1 debasis debasis    75 Nov 18 11:13 a.sh
-rw-r--r-- 1 debasis debasis    39 Nov 30 16:46 a.txt
-rw-r--r-- 1 debasis debasis    75 Nov 13 10:46 a2.sh
-rw-r--r-- 1 debasis debasis    70 Nov 13 10:29 aa.sh
-rwxr-xr-x 1 debasis debasis    20 Dec  2 16:16 aa.txt
-rw-r--r-- 1 debasis debasis   188 Nov 30 16:12 abcd.txt
```

Example: Display line numbers in a file:

[std@server05 ~]$ cat -n prime.sh

```
#!/bin/bash
 2  echo enetr a number
 3  read n
 4  flag=1
 5  i=2
 6  while [ $i -lt $n ]
 7  do
 8  r=`expr $n % $i`
 9  if [ $r -eq 0 ]
10  then
11  flag=0
12  break
13  fi
14  i=`expr $i + 1`
15  done
16  if [ $flag -eq 1 ]
17  then
18  echo $n is a prime
19  else
20  echo $n is not a prime
```

```
21  fi
22
```

Example: Display line numbers from 7 to 8 in a file:

M=8

N=7

Tail=8-7+1=1+1=2

[std@server05 ~]$ head -n 8 prime.sh | tail -2

```
do
r=`expr $n % $i`
```

```
debasis@LAPTOP-H3N6JCNE:~$ cat -n cc.txt
     1      December 2020
     2   Su Mo Tu We Th Fr Sa
     3            1  2  3  4  5
     4    6  7  8  9 10 11 12
     5   13 14 15 16 17 18 19
     6   20 21 22 23 24 25 26
     7   27 28 29 30 31
     8
```

Q:Display lines numbers from calendar 3 to 5

```
debasis@LAPTOP-H3N6JCNE:~$ cal | head -n 5 | tail -n 3
```

output:

```
       1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
```

Example: Display line numbers from 3 to 8 in a file:

M=8

N=3

Tail=8-3+1=5+1=6

[std@server05 ~]$ head -n 8 prime.sh | tail –n 6

```
read n
flag=1
i=2
while [ $i -lt $n ]
do
r=`expr $n % $i`
```

Example: Display line numbers from 3 to 8 in a file:
M=8
N=3
Tail=8-3+1=5+1=6

[std@server05 ~]$ head -8 prime.sh | tail -6

```
read n
flag=1
i=2
while [ $i -lt $n ]
do
r=`expr $n % $i`
```

Example: Display line numbers from 4 to 8 in a file:
M=8
N=4
Tail=m-n+1=8-4+1=5

[std@server ~]$ head -n 8 prime.sh | tail -n 5

```
flag=1
i=2
while [ $i -lt $n ]
do
r=`expr $n % $i`
```

```
[std1719bca15@server05 ~]$ cat -n prime.sh

     1  #!/bin/bash
     2  echo enetr a number
     3  read n
     4  flag=1
     5  i=2
     6  while [ $i -lt $n ]
     7  do
     8  r=`expr $n % $i`
     9  if [ $r -eq 0 ]
    10  then
    11  flag=0
    12  break
```

```
13  fi
14  i=`expr $i + 1`
15  done
16  if [ $flag -eq 1 ]
17  then
18  echo $n is a prime
19  else
20  echo $n is not a prime
21  fi
22
```

Example: Display line numbers from 11 to 20 in a file:

M=20
N=11
TAIL=M-N+1=20-11+1=10

[std@server~]$ head -n 20 prime.sh | tail -n 10

```
flag=0
break
fi
i=`expr $i + 1`
done
if [ $flag -eq 1 ]
then
echo $n is a prime
else
echo $n is not a prime
```

Example: Display line numbers from 13 to 20 in a file:

M=20
N=13
TAIL=M-N+1=20-13+1=7+1=8

[std@server]$ head -n 20 prime.sh | tail -n 8

```
fi
i=`expr $i + 1`
done
if [ $flag -eq 1 ]
then
echo $n is a prime
else
echo $n is not a prime
[std1719bca15@server05 ~]$
```

How to use the head with pipeline(|): The head command can be piped with other commands. In the following example, the output of the ls command is piped to head to show only the three most recently modified files or folders.

```
12. Display all recently modified or recently used files.


13.
14. $ ls –t

15. e.txt
16. d.txt
17. c.txt
18. b.txt
19. a.txt
20.
21. Cut three most recently used file.

22. $ ls -t | head -n 3


23.
24.
25. e.txt
26. d.txt
27. c.txt
28.
```

It can also be piped with one or more filters for additional processing. For example, the sort filter could be used to sort the three most recently used files or folders in the alphabetic order.

```
$ ls -t | head -n 3 | sort

c.txt
d.txt
e.txt
```

Vertical division of file (cut):

cut command :

The cut command in UNIX is a command for cutting out the sections from each line of files and writing the result to standard output. It can be used to cut parts of a line by **byte position, character and field**. Basically the cut command slices a line and extracts the text. It is necessary to specify option with command otherwise it gives error.

-c (column): To cut by character use the -c option. This selects the characters given to the -c option. This can be a list of numbers separated comma or a range of numbers separated by hyphen(-). **Tabs and backspaces** are treated as a character. It is necessary to specify list of character numbers otherwise it gives error with this option.

```
debasis@LAPTOP-H3N6JCNE:~$ date
Fri Dec 18 14:56:09 IST 2020
debasis@LAPTOP-H3N6JCNE:~$ date | cut -c 1-3
Fri
debasis@LAPTOP-H3N6JCNE:~$ date | cut -c 1-9,10-16
Fri Dec 18 14:58
```

```
debasis@LAPTOP-H3N6JCNE:~$ echo "MADAM" | cut -c 1
M
debasis@LAPTOP-H3N6JCNE:~$ echo "MADAM" | cut -c 5
M
debasis@LAPTOP-H3N6JCNE:~$ echo "MADAM" | cut -c 2
A
debasis@LAPTOP-H3N6JCNE:~$ echo "MADAM" | cut -c 4
A
```

```
#Check input string is PALINDROME OR NOT
#!/bin/bash
echo -n "Enter String: "
read string
len=`expr length $string`
echo "Length is : " $len
flag=1
i=1
while [ $i -le $len ]
do
    x=`expr $string | cut -c $i`
    y=`expr $string | cut -c $len`
    if [ $x != $y ]
    then
        flag=0
        break
    fi
    i=`expr $i + 1`
    len=`expr $len - 1`
done
if [ $flag -eq 1 ]
then
    echo $string is PALINDROME
```

```
else
      echo $string is NOT a Palindrome
fi
echo “Job Over.”
output:
debasis@LAPTOP-H3N6JCNE:~$ sh palinstring.sh
Enter String: madam
Length is :  5
madam is PALINDROME
Job Over
```

```
#count total number of vowels in a word
echo -n "Enter string: "
read string
len=`expr length $string`
echo "Total Lenght: " $len
v=0
i=1
while [ $i -le $len ]
do
      m=`expr $string | cut -c $i`
      if [ $m = 'a' -o $m = 'e' -o $m = 'i' -o $m = 'o' -o $m = 'u' ]
      then
            v=`expr $v + 1`
      fi
i=`expr $i + 1`
done
echo "Total vowels: "$v
output:
debasis@LAPTOP-H3N6JCNE:~$ sh vwl.sh
Enter string: student
Total Lenght:  7
Total vowels: 2
```

Cut:
-f (field): -c option is useful for fixed-length lines. Most unix files doesn’t have fixed-length lines. To extract the useful information you need to cut by fields rather than columns. List of the fields number specified must be separated by comma. *Ranges are not described with -f option*. **cut** uses **tab** as a default field delimiter but can also work with other delimiter by using **-d** option.

Syntax:

```
$cut -d "delimiter" -f (field number) filename
```

If -d option is used then it considered space as a field separator or delimiter:

```
debasis@LAPTOP-H3N6JCNE:~$ date
Fri Dec 18 14:46:42 IST 2020
```

```
debasis@LAPTOP-H3N6JCNE:~$ date | cut -d " " -f 1-3,6
Fri Dec 18 2020
```

```
Total lines/users:
debasis@LAPTOP-H3N6JCNE:~$ ls -l | cut -d " " -f3 | wc -l
89
```

```
$ cut -d " " -f1 state.txt
Andhra
Arunachal
Assam
Bihar
Chhattisgarh
```

debasis@LAPTOP-H3N6JCNE:~/$ cat uu.txt
NAME,AGE,CLASS
xx,20,1st
yy,21,2nd
zz,22,3rd

debasis@LAPTOP-H3N6JCNE:~$ cat uu.txt | cut -d "," -f 1,3
NAME,CLASS
xx,1st
yy,2nd
zz,3rd

debasis@LAPTOP-H3N6JCNE:~$ date

Fri Dec 18 16:02:58 IST 2020

debasis@LAPTOP-H3N6JCNE:~$ date | cut -d " " -f 3

18

debasis@LAPTOP-H3N6JCNE:~$ date | cut -d " " -f -3

Fri Dec 18 [output from 1 to 3 fields]

debasis@LAPTOP-H3N6JCNE:~$ date | cut -d " " -f 1-3

Fri Dec 18

debasis@LAPTOP-H3N6JCNE:~$ date | cut -d " " -f -5

Fri Dec 18 16:03:21 IST [output from 1 to 5 fields]

Applications of cut Command

1. How to use tail with pipes(|): The cut command can be piped with many other commands of the unix. In the following example output of the **cat** command is given as input to the **cut** command with **-f** option to sort the state names coming from file state.txt in the reverse order.

```
$ cat state.txt | cut -d ' ' -f 1 | sort –r
```

```
Chhattisgarh
Bihar
Assam
Arunachal
Andhra
```

```
#Example of for LOOP
#!/bin/bash

for n in 1 2 3 4 5
 do
         echo $n
 done
echo job over
output:
debasis@LAPTOP-H3N6JCNE:~$ sh ff.sh
1
2
3
4
5
job over_____________
```

```
#Example of for LOOP
#!/bin/bash
cnt=0
```

```
for w in India is Great
 do
         echo $w
         cnt=`expr $cnt + 1`
 done
 echo "Total words: " $cnt
echo job over
```

output:

```
debasis@LAPTOP-H3N6JCNE:~$ sh fff.sh
India
is
Great
Total words:  3
job over
```

```
#using case and advance for loop
#count total no of vowels and words from
given input file
#!/bin/bash
echo -n "Enter filename: "
read filename
cnt=0
v=0
for w in `cat $filename`
do
        echo $w
        i=1
        len=`expr length $w`
        while [ $i -le $len ]
        do
         m=`expr $w | cut -c $i`
        case $m in
        [aeiouAEIOU])v=`expr $v + 1`
                                ;;
```

```
        esac
         i=`expr $i + 1`
       done
        cnt=`expr $cnt + 1`

done
echo "Total word: "  $cnt
echo "Total vowels: " $v
output:
debasis@LAPTOP-H3N6JCNE:~$ sh ff4.sh
Enter filename: a55.txt
this
is
my
copied
files
job
over
Total word:  7
Total vowels:  10
```

```
#count digits , vowels and other characters from user input file

#!/bin/bash

echo -n "Enter filename: "

read filename

d=0

v=0

c=0

for string in `cat $filename`
```

```
do
len=`expr length $string`
#echo "Total Length= " $len
i=1
while [ $i -le $len ]
do
      x=`expr $string | cut -c $i`
      case $x in
            [0-9])d=`expr $d + 1`
                  ;;
            [aeiouAEIOU]) v=`expr $v + 1`
                 ;;
            *)c=`expr $c + 1`
      esac
i=`expr $i + 1`
done
done
echo "Total digits: " $d
echo "Total vowels: :" $v
echo "Total other chacters : " $c

echo "Job over."
output:
debasis@LAPTOP-H3N6JCNE:~$ sh filecount.sh
Enter filename: a.txt
Total digits:  2
Total vowels: : 11
```

```
Total other chacters :  21

Job over.
```

```
#Print Appropriate Message on time
#!/bin/bash

date>filename1
cat filename1
cut -d " " -f 4 filename1>filename2
cat filename2
time=`cut -d ":" -f1 filename2`
echo $time
if [ $time -lt 12 ]
then
        echo "Good Morning."
elif [ $time -lt 16 ]
then
        echo "Good Afternoon."
else
        echo "Good Evening."
 fi
```

output:

```
debasis@LAPTOP-H3N6JCNE:~$ sh timeMsg.sh
Fri Dec 18 15:48:52 IST 2020
15:48:52
15
Good Afternoon.
```

Paste command in Linux with examples:

Paste files (paste):

Paste command is one of the useful commands in Unix or Linux operating system. It is used to **join files horizontally (parallel merging) by outputting lines consisting of lines from each file specified, separated by** tab **as delimiter, to the standard output.**

NAME

paste - merge lines of files

Syntax:

```
paste [OPTION]... [FILES]...
```

Options:

1. -d (delimiter): Paste command uses the tab delimiter by default for merging the files.

```
-d, --delimiters=LIST
                reuse characters from LIST
instead of TABs
```

2. -s (serial): We can merge the files in sequentially manner using the -s option.

```
        -s, --serial
                paste one file at a time
instead of in parallel
```

```
debasis@LAPTOP-H3N6JCNE:~$ cat num.txt
1
2
3
4
5
debasis@LAPTOP-H3N6JCNE:~$ cat state.txt
```

```
West Bengal
Tripura
Bihar
Jharkhand
Uttarpradesh

debasis@LAPTOP-H3N6JCNE:~$ cat capital.txt
Kolkata
Agartala
Patna
Ranchi
Lucknow
```

```
debasis@LAPTOP-H3N6JCNE:~$ paste num.txt
state.txt capital.txt
1       West Bengal     Kolkata
2       Tripura         Agartala
3       Bihar           Patna
4       Jharkhand       Ranchi
5       Uttarpradesh    Lucknow
```

```
debasis@LAPTOP-H3N6JCNE:~$ paste -d "|"
num.txt state.txt capital.txt
1|West Bengal|Kolkata
2|Tripura|Agartala
```

```
3|Bihar|Patna
4|Jharkhand|Ranchi
5|Uttarpradesh|Lucknow
```

```
debasis@LAPTOP-H3N6JCNE:~$ paste -d "|," num.txt state.txt capital.txt
1|West Bengal,Kolkata
2|Tripura,Agartala
3|Bihar,Patna
4|Jharkhand,Ranchi
5|Uttarpradesh,Lucknow
```

```
debasis@LAPTOP-H3N6JCNE:~$ paste -s  num.txt state.txt capital.txt
1               2       3       4             5
West Bengal     Tripura     Bihar   Jharkhand  Uttarpradesh
Kolkata          Agartala   Patna   Ranchi      Lucknow
```

```
debasis@LAPTOP-H3N6JCNE:~$ paste -s -d "|" num.txt state.txt capital.txt
1|2|3|4|5
West Bengal|Tripura|Bihar|Jharkhand|Uttarpradesh
```

```
Kolkata|Agartala|Patna|Ranchi|Lucknow
```

SORT command in Linux/Unix with examples:

SORT command is used to sort a file, arranging the records in a particular order. By default, the sort command sorts file assuming the contents are ASCII. Using options in sort command, it can also be used to sort numerically.

- SORT command sorts the contents of a text file, line by line.
- sort is a standard command line program that prints the lines of its input or concatenation of all files listed in its argument list in sorted order.
- The sort command is a command line utility for sorting lines of text files. It supports sorting alphabetically, in reverse order, by number, by month and can also remove duplicates.

NAME

sort - sort lines of text files

SYNOPSIS / Syntax :

sort [OPTION]... [FILE]...

-n, --numeric-sort

compare according to string numerical value

-r, --reverse

reverse the result of comparisons

-k, --key=KEYDEF

sort via a key; KEYDEF gives location and type(COLUMN wise)

-t, --field-separator=SEP

use SEP instead of non-blank to blank transition

-M, --month-sort

compare (unknown) < 'JAN' < ... < 'DEC'

-u, --unique

```
debasis@LAPTOP-H3N6JCNE:~$ cat employee.txt
NAME AGE Salary
RAJA  35  35000
RAHUL 42  42000
MONOJ 40  40000
ALOK  25  20000
```

```
debasis@LAPTOP-H3N6JCNE:~$ sort employee.txt
ALOK  25  20000
MONOJ 40  40000
NAME AGE Salary
RAHUL 42  42000
RAJA  35  35000
```

-r Option: Sorting In Reverse Order / descending order:

```
debasis@LAPTOP-H3N6JCNE:~$ sort -r employee.txt
RAJA  35  35000
RAHUL 42  42000
NAME AGE Salary
MONOJ 40  40000
ALOK  25  20000
```

Option : Unix provides the feature of sorting a table on the **basis of any column number**
For example, use "-k 2" to sort on the second column.

```
debasis@LAPTOP-H3N6JCNE:~$ sort -k 2 employee.txt
ALOK  25  20000
RAJA  35  35000
MONOJ 40  40000
RAHUL 42  42000
NAME AGE Salary
```

debasis@LAPTOP-H3N6JCNE:~$ **sort -k 2 -r employee.txt**
NAME AGE Salary
RAHUL 42 42000
MONOJ 40 40000
RAJA 35 35000
ALOK 25 20000

debasis@LAPTOP-H3N6JCNE:~$ **cat employee.txt | tr -s " " "|" >emp.txt**

```
debasis@LAPTOP-H3N6JCNE:~$ cat emp.txt
NAME|AGE|Salary
RAJA|35|35000
RAHUL|42|42000
MONOJ|40|40000
ALOK|25|20000
```

```
debasis@LAPTOP-H3N6JCNE:~$ cat emp.txt
NAME|AGE|Salary
RAJA|35|35000
RAHUL|42|42000
MONOJ|40|40000
ALOK|25|20000
```

sort -t "SEPARATOR" : Use the provided separator to identify the fields.

The default separator for fields is the space character. The -t option can be used to change the separator.

```
debasis@LAPTOP-H3N6JCNE:~$ sort -t "|" -k 3 emp.txt
ALOK|25|20000
RAJA|35|35000
MONOJ|40|40000
RAHUL|42|42000
NAME|AGE|Salary
```

-n Option : To sort a file numerically
-nr option : To sort a file with **numeric data in reverse order**

```
debasis@LAPTOP-H3N6JCNE:~$ sort -t "|" -k 3 -nr emp.txt
```

```
RAHUL|42|42000

MONOJ|40|40000

RAJA|35|35000

ALOK|25|20000

NAME|AGE|Salary
```

```
debasis@LAPTOP-H3N6JCNE:~$ sort dupsrt.txt

12

12

35

40

50

ab

cd

ef

xz

zx
```

```
-u for unique elements

debasis@LAPTOP-H3N6JCNE:~$ sort -u dupsrt.txt

12

35

40

50

ab
```

```
cd

ef

xz

zx
```

```
debasis@LAPTOP-H3N6JCNE:~$ cat dupsrt.txt

12

35

40

50

12

JAN

FEB

MARCH

APR

MAY

JUNE

ab

cd

ef

xz

zx
```

```
#Sort month …JAN<FEB…….<DEC

debasis@LAPTOP-H3N6JCNE:~$ sort -M dupsrt.txt
```

```
12
12
35
40
50
ab
cd
ef
xz
zx
JAN
FEB
MARCH
APR
MAY
JUNE
```

```
#sort month in reverse order
debasis@LAPTOP-H3N6JCNE:~$ sort -M -r dupsrt.txt
JUNE
MAY
APR
MARCH
FEB
JAN
```

zx

xz

ef

cd

ab

50

40

35

12

12

uniq command in unix:

Finding repetition and non- repetition (uniq):

The linux **uniq** command is basically used to remove all repeated lines in a file. This command is used when a line is repeated multiple times and replace these multiple lines with one line. This command is designed to work on **sorted** files.

Purpose : Remove repetitious lines from the sorted 'input-file' and send unique lines to the 'output-file'. If no 'output-file' is specified, the output of the command is send to standard output. if no 'input-file' is specified, the command takes input from standard output.

```
NAME
       uniq - report or omit repeated lines
```

Syntax / `SYNOPSIS :`

```
       uniq [OPTION]... [INPUT [OUTPUT]]

DESCRIPTION:
       Filter adjacent matching lines from INPUT (or standard input),
writing to OUTPUT (or standard output).

       -c, --count
```

```
       prefix lines by the number of occurrences

-d, --repeated
       only print duplicate lines, one for each group

-D     print all duplicate lines
-i, --ignore-case
       ignore differences in case when comparing

 -u, --unique
       only print unique lines
```

Examples :

```
debasis@LAPTOP-H3N6JCNE:~$ cat>aaa.txt
This
This
is
my
file
file
```

```
debasis@LAPTOP-H3N6JCNE:~$ uniq aaa.txt bbb.txt
debasis@LAPTOP-H3N6JCNE:~$ cat bbb.txt
This
is
my
file
```

```
debasis@LAPTOP-H3N6JCNE:~$ uniq aaa.txt
This

is

my

file
```

```
-c : prefix lines by the number of occurrences
debasis@LAPTOP-H3N6JCNE:~$ uniq -c aaa.txt
```

2 This

1 is

1 my

2 file

```
-D : print all duplicate lines
```

debasis@LAPTOP-H3N6JCNE:~$ uniq -D aaa.txt

This

This

file

file

```
-d : only print duplicate lines, one for each group
```

debasis@LAPTOP-H3N6JCNE:~$ uniq -d aaa.txt

This

file

```
 -u : only print unique lines
```

debasis@LAPTOP-H3N6JCNE:~$ uniq -u aaa.txt

is

my

debasis@LAPTOP-H3N6JCNE:~$ ls -l | cut -d " " -f3 | uniq

output:

debasis

COMMAND: tr- Manipulating characters :

NAME

tr - translate or delete characters

SYNOPSIS

tr [OPTION]... SET1 [SET2]

DESCRIPTION

Translate, squeeze, and/or delete characters from standard input, writing to standard output.

-c, -C, --complement

use the complement of SET1

-d, --delete

delete characters in SET1, do not translate

-s, --squeeze-repeats

replace each sequence of a repeated character that is listed in the last specified SET, with a single occurrence of that character

```
debasis@LAPTOP-H3N6JCNE:~$ date | tr -d 'F'

ri Dec 18 19:17:12 IST 2020
```

```
debasis@LAPTOP-H3N6JCNE:~$ date | tr -d 'Fr'

i Dec 18 19:17:45 IST 2020

debasis@LAPTOP-H3N6JCNE:~$ date | tr -d 'Fri'
```

6. To remove all the digits from the string, use command:

Example:

```
$ echo "my ID is 73535" | tr -d [:digit:]
```

```
debasis@LAPTOP-H3N6JCNE:~$ date | tr -d '[:digit:]'

Fri Dec  :: IST
```

Example:

debasis@LAPTOP-H3N6JCNE:~$ date | tr -d [:digit:]

Wed Jan :: IST

Example:

debasis@LAPTOP-H3N6JCNE:~$ date | tr -cd [:digit:]

output: 61051282021 #complement of digits ,use option -cd

To REMOVE Characters:

debasis@LAPTOP-H3N6JCNE:~$ date

Wed Jan 6 10:47:18 IST 2021

debasis@LAPTOP-H3N6JCNE:~$ date | tr -d '[a-z][A-Z]'

6 10:47:23 2021

debasis@LAPTOP-H3N6JCNE:~$ date | tr -d [a-z][A-Z]

6 10:47:32 2021

cd : complement of digits-

How to complement the sets using -c option
You can complement the SET1 using -c option. For example, to remove all characters except digits, you can use the following

debasis@LAPTOP-H3N6JCNE:~$ date | tr -cd [:digit:]

181956592020

debasis@LAPTOP-H3N6JCNE:~$ cat a.txt

today is Monday.

This is testing file.

debasis@LAPTOP-H3N6JCNE:~$ cat a.txt | tr [:lower:] [:upper:]

TODAY IS MONDAY.

THIS IS TESTING FILE.

debasis@LAPTOP-H3N6JCNE:~$ cat a.txt | tr '[:lower:]' '[:upper:]'

TODAY IS MONDAY.

THIS IS TESTING FILE.

debasis@LAPTOP-H3N6JCNE:~$ cat a.txt

today is Monday.

This is testing file.

Example: To delete characters T or h or s

debasis@LAPTOP-H3N6JCNE:~$ cat a.txt | tr -d 'This'

today Monday.

** tetng fle.**

debasis@LAPTOP-H3N6JCNE:~$ cat a.txt | tr -d 'This' | tr '[a-z]' '[A-Z]'

TODAY MONDAY.

** TETNG FLE.**

debasis@LAPTOP-H3N6JCNE:~$ cat a.txt
today is Monday.
This is testing file.

debasis@LAPTOP-H3N6JCNE:~$ cat a.txt | tr -d 'tis'
oday Monday.
Th eng fle.

debasis@LAPTOP-H3N6JCNE:~$ ls -l | tr -s " " "|"

debasis@LAPTOP-H3N6JCNE:~$ ls -l | tr -s " " "|" | cut -d "|" -f 3

```
debasis@LAPTOP-H3N6JCNE:~$ cat a.txt

today is Monday.

This is testing file.

debasis@LAPTOP-H3N6JCNE:~$ tr "[a-z]" "[A-Z]" <a.txt

TODAY IS MONDAY.

THIS IS TESTING FILE.

debasis@LAPTOP-H3N6JCNE:~$ tr "[a-z]" "[A-Z]" < a.txt  > aa.txt

debasis@LAPTOP-H3N6JCNE:~$ cat aa.txt

TODAY IS MONDAY.

THIS IS TESTING FILE.

debasis@LAPTOP-H3N6JCNE:~$ cat a.txt

today is Monday.

This is testing file.
```

grep command in Unix/Linux:

searching pattern using grep:

The grep filter searches a file for a particular pattern of characters, and displays all lines that contain that pattern. The pattern that is searched in the file is referred to as the regular expression (grep stands for globally search for regular expression and print out).
Syntax:

```
grep [options] pattern [files]
```

```
Options Description
-c : This prints only a count of the lines that match a pattern

-i : Ignores, case for matching

-n : Display the matched lines and their line numbers.

-v : This prints out all the lines that do not matches the pattern
```

Show line number while displaying the output using grep -n : To show the line number of file with the line matched.

```
$ grep -n "unix" filename.txt
```

Output:

```
1:unix is great os. unix is opensource. unix is free os.
4:uNix is easy to learn.unix is a multiuser os.Learn unix .unix is a
powerful.
```

2. Inverting the pattern match : You can display the lines that are not matched with the specified search sting pattern using the -v option.

```
$ grep -v "unix" filename.txt
```

Output:

```
learn operating system.
Unix linux which one you choose.
```

Matching the lines that start with a string :

The ^ regular expression pattern specifies the start of a line. This can be used in grep to match the lines which start with the given string or pattern.

```
$ grep "^unix" filename.txt
```

Output:

```
unix is great os. unix is opensource. unix is free os.
```

Matching the lines that end with a string :

The **$** regular expression pattern specifies the end of a line. This can be used in grep to match the lines which end with the given string or pattern.

```
$ grep "os$" filename.txt
```

#check whether a given input user is VALID user or NOT

#!/bin/bash

echo -n "Enter user name: "

read user

if grep $user /etc/passwd >/dev/null

then

```
        echo $user is a VALID user
else
        echo $user is NOT a valid user
fi
out:
debasis@LAPTOP-H3N6JCNE:~$ sh user.sh
Enter user name: root
root is a VALID user
debasis@LAPTOP-H3N6JCNE:~$ sh user.sh
Enter user name: ROOT
ROOT is NOT a valid user
```

Anchors:

Anchors

The caret (^) and dollar sign ($) characters are treated as *anchors* in regular expressions. This means that they cause the match to occur only if the regular expression is found at the beginning of the line (**^**) or at the end of the line (**$**) ,respectively.

Example: Displays lines starting with the vivek only:

```
$ grep ^vivek /etc/passwd
```

Example: Display only lines starting with the word vivek only i.e. do not display vivekgite, vivekg etc:

```
$ grep -w ^vivek /etc/passwd
```

Example: Find lines ending with word foo:

```
$ grep 'foo$' filename
```

Example: search for blank lines with the following examples:

```
$ grep '^$' filename
```

How to match sets of character using grep:

The dot (.) matches any single character. You can match specific characters and character ranges using [..] syntax.

Example: Say you want to Match both 'Vivek' or 'vivek':

```
$grep '[vV]ivek' filename
```

OR

```
$grep '[vV][iI][Vv][Ee][kK]' filename
```

Example:Let us match digits and upper and lower case characters. For example, try to math words such as vivek1, Vivek2 and …so on:

```
$grep -w '[vV]ivek[0-9]' filename
```

Within a bracket expression, the name of a character class enclosed in "[:" and ":]" stands for the list of all characters belonging to that class. Standard character class names are:

- `[[:alnum:]]` – Alphanumeric characters [a-z A-Z 0-9].
- `[[:alpha:]]` – Alphabetic characters
- `[[:blank:]]` – Blank characters: space and tab.
- `[[:digit:]]` – Digits: '0 1 2 3 4 5 6 7 8 9'.
- `[[:lower:]]` – Lower-case letters: 'a b c d e f g h i j k l m n o p q r s t u v w x y z'.
- `[[:space:]]` – Space characters: tab, newline, vertical tab, form feed, carriage return, and space.
- `[[:upper:]]` – Upper-case letters: 'A B C D E F G H I J K L M N O P Q R S T U V W X Y Z'.
- **[[:xdigit:]] -** Hex digits [0-9 a-f A-F]

Example:
In this example match all upper case letters:

```
grep '[:upper:]' filename
```

MAKAUT Syllabus:Brief idea of using Basic Regular Expression (BRE), Extended Regular Expression (ERE), and egrep, grep:

Regular Expressions in grep:

Regular Expressions(RE) is nothing but a pattern to match for each input line. A pattern is a sequence of characters.

Two types of regex:

The grep understands three different types of regular expression syntax as follows:

1. Basic (BRE)
2. Extended (ERE)

Basic Regular Expression(BRE):

Regular Expression provides an ability to match a “string of text” in a very flexible and concise manner. A “string of text” can be further defined as a single character, word, sentence or particular pattern of characters.

Like the shell’s wild–cards which match similar filenames with a single expression, grep uses an expression of a different sort to match a group of similar patterns.

1.[abc]: Matches any one of a set characters

Example1: $grep “New[abc]” filename

Output: It specifies the search pattern as :

```
Newa , Newb or Newc
```

Example2: $grep “[aA]g[ar][ar]wal” filename

Output: It specifies the search pattern as

```
Agarwal , Agaawal , Agrawal , Agrrwal
agarwal , agaawal , agrawal , agrrwal
```

2.[a-z] with hyphen: Matches any one of a range characters
Example-1: `$grep "New[a-e]" filename`
It specifies the search pattern as : `Newa , Newb or Newc , Newd, Newe`

`Example-2: $grep "New[0-9][a-z]" filename`

3 .**^**: The pattern following it must occur at the beginning of each line
4. **^ with [ ] :** The pattern must not contain any character in the set specified
Use ^ with [ ]: The pattern must not contain any character in the set specified
`Example-1: $grep "New[^a-c]" filename`
Output: It specifies the pattern containing the word "New" followed by any character other than an 'a','b', or 'c'

5.**$**: The pattern preceding it must occur at the end of each line
`$ grep "come$" file.txt`

6.*: zero or more occurrences of the previous character
`$ grep "[aA]gg*[ar][ar]wal" file.txt`

grep Regular Expressions Examples:

Example-1: Search for 'vivek' in /etc/passswd

```
$ grep 'vivek' /etc/passwd
```

Example-2: Search vivek or raj in any case

```
$ grep -E -i -w 'vivek|raj' /etc/passwd
```

The PATTERN in last example, used as an extended regular expression.

The following will match word Linux or UNIX in any case:

```
$ egrep -i '^(linux|unix)' filename
```

Linux grep vs egrep command:

The egrep is the same as **`grep -E`** command. It interpret PATTERN as an extended regular expression.

`egrep` is an acronym for "**Extended Global Regular Expressions Print**". It is a program which scans a specified file line by line, returning lines that contain a pattern matching a given regular expression.

```
debasis@LAPTOP-H3N6JCNE:~/search$ cat file.txt

i am a student of BCA

unix or linux is more useful os in today's technology



job BCA over

bcabatch msit
```

```
debasis@LAPTOP-H3N6JCNE:~/search$ grep -E -i -n  "bca|msit" file.txt

1:i am a student of BCA

4:job BCA over

5:bcabatch msit

debasis@LAPTOP-H3N6JCNE:~/search$ grep  -i -n  "bca|msit" file.txt

debasis@LAPTOP-H3N6JCNE:~/search$ egrep  -i -n  "bca|msit" file.txt

1:i am a student of BCA

4:job BCA over

5:bcabatch msit
```

```
debasis@LAPTOP-H3N6JCNE:~$ ls -l | egrep -n -c 'Dec|CSE'

64

debasis@LAPTOP-H3N6JCNE:~$ ls -l | grep -E -n -c 'Dec|CSE'

64
```

MAKAUT 2019-20

Q:8(a) A file (file 1) contains the following Patterns:

Agarwal, Agarwal, Aggarwal and Aggarwal.

How to search all of them using BRE character subset ? How to search metacharacters in a file ?

Hints:

Example2: `$grep "[aA]g[ar][ar]wal" filename`

Output: It specifies the search pattern as

```
Agarwal , Agaawal , Agrawal , Agrrwal
agarwal , agaawal , agrawal , agrrwal
```

(b) Briefly explain grep-E with suitable examples.

(c) Write the commands to do the following using grep:

(i) Search the string "HELLO" from the file file 1 if it occurs at the very beginning of a line.
(ii) Search the string "HELLO" from the file file 1 if it occurs at the end of a line.
(iii) Search the string "HELLO" from the file file 1 if it is the only word in the line.

Multiple choice:

Q: Which of the following expression is a correct wildcard pattern if we want an expression in which the last character is not numeric?

(a) *[!0] (b) *[0-9]

(c) [0-9] (d) *[!0-9]

Q: Write a shell script to check whether a given input user is VALID ot NOT.

Q: Write a shell script to check whether a given input user is Logged in or NOT.