Unit-2 Linux Commands.

Introduction:

- We know that, Linux operating system provide us two environments to work with linux kernel. One is Shell and other is desktop.
- Also, most of work in linux done by shell which is command line interpreter that took textual based command as input and interpret into binary format give it to kernel for processing.
- Hence, to work on shell we have to learn different commands. Linux have different categories of commands which are as follow:
  1) File and directory management Commands.
  2) File and directory operation commands.
  3) Filter commands.
  4) Communication commands.
  5) 5) Archive and File Compression Commands
  6) Let's see these categories of commands in details:

I) File and Directory Management Commands:

- The commands which are used to manage (Create, remove, rename, move etc.) file and directory are called 'File and directory management commands'.
- Linux provides several file and directory management commands which are discussed below-

1) mkdir command: (Make Directory)

This command is used to create new directory (or directories) at current working location. (If it is not already exist)

Syntax: mkdir OPTION Directory1 Directory2 Directory3

Here, at OPTION we can use:

-m (MODE): This option is used to set file mode to rwx (same as in chmod)

-p (parent): This option is used to make a parent directory (nesting of subdirectories) as needed (intervening parent directories)

-v (verbose): This option is used to print appropriate alert message for each created directory.

Examples:

```
Rps@linux-j7rs:~> mkdir -v    grandFather
mkdir: created directory 'grandFather'

Rps@linux-j7rs:~>mkdir    -v       aa        bb          cc
mkdir: created directory 'aa'
mkdir: created directory 'bb'
mkdir: created directory 'cc'

Rps@linux-j7rs:~>mkdir    -v       -p      Father/Son/Daughter
mkdir: created directory 'Father'
mkdir: created directory 'Father/Son'
mkdir: created directory 'Father/Son/Daughter'

Rps@linux-j7rs:~>mkdir    -v      grandFather
mkdir: cannot create directory 'grandFather': File exists
Rps@linux-j7rs:~>mkdir     -v      -m        u+r-wx       flower
mkdir: created directory 'flower'
```

***Note:** Here, 'flower' directory is created with <u>add owner read</u> permission and <u>remove owner write and execute</u> permission. See result by <u>ls –l</u> command.*

```
Rps@linux-j7rs:~>mkdir    -v      -m     a=rx    rose
mkdir: created directory 'rose'
```

***Note:** Here, 'rose' directory is created with <u>read and execute permission to all.</u> See result by <u>ls –l</u> command.*

2) cd command: (Change Directory)

- This command is used to change the current working directory.
- We can use cd command by following way:
  - cd without any directory name changes back to the home directory of user.
  - cd with single dot (.) that refer to your current working directory.
  - cd with double dot (..) that changes to parent directory of pwd.
  - cd with directory name that changes to specified directory. And it has following syntax-

Syntax :　　cd　　DirectoryName

Examples:

```
Rps@linux-j7rs:/home/sangola/Desktop/myFolder>  cd
Rps@linux-j7rs: ~ >

Rps@linux-j7rs: ~ > cd .
Rps@linux-j7rs: ~ >

Rps@linux-j7rs: ~ > cd home
Rps@linux-j7rs:/home >

Rps@linux-j7rs: /home> cd sangola
Rps@linux-j7rs:/home/sangola >

Rps@linux-j7rs: /home/sangola> cd Desktop
Rps@linux-j7rs:/home/sangola/Desktop >

Rps@linux-j7rs: /home/sangola/Desktop> cd ..
Rps@linux-j7rs:/home/sangola>

Rps@linux-j7rs: /home/sangola> cd ..
Rps@linux-j7rs:/home>

Rps@linux-j7rs: /home> cd sangola/Desktop
Rps@linux-j7rs:/home/sangola/Desktop>
```

3) rmdir command: (Remove Directory)

This command is used to remove the directory (or Directories) if they are empty.

Note: If directory is not empty and we try to remove it then it will not remove.

Syntax:　　rmdir　　OPTION　Directory1　　Directory2　Directory3

Here, at OPTION we can use:

-p (parents): This option is used to remove DIRECTORY and its ancestors

-v (verbose): This option is used to print appropriate alert message for each processed directory.

Examples:

```
Rps@linux-j7rs:~>rmdir  -v   grandFather
rmdir: removing directory, ‘grandFather’
Rps@linux-j7rs:~>rmdir    -v      aa        bb          cc
rmdir:  removing  directory,  ‘aa’
rmdir:  removing  directory,  ‘bb’
rmdir: removing directory,  ‘cc’
Rps@linux-j7rs:~>rmdir    -v      -p      Father/Son/Daughter
rmdir: removing directory, ‘Father’
rmdir: removing directory , ‘Father/Son’
rmdir: removing directory , ‘Father/Son/Daughter’
Rps@linux-j7rs:~>rmdir    -v     grandfather
rmdir: Failed to remove ‘grandFather’: No such file or directory
```

4) pwd command: (Present Working Directory)

- This command is <u>used to display or print name of current or present working directory where you work</u>.

Syntax: **pwd**

Examples:

```
Rps@linux-j7rs:/home/sangola/Desktop>  pwd
 /home/sangola/Desktop
```

5) file command:

- This command is <u>used to display all files contained in the directory along with type of file</u>.
- In short, file command <u>determines files type contained in the directory</u>.

Syntax: 1) file *

In above syntax, file command determines and display <u>all files (*)</u> along with type of file contained in the pwd.

2) file fileOrDirectory

In above syntax, file command <u>determines type</u> of mentioned file.

Examples:

```
Rps@linux-j7rs:/home/sangola/Desktop>  file    *
myFolder:                       directory
GnomeOnlineHelp.desktop: UTF-8 Unicode text
SuSE.desktop:                   ASCII text

Rps@linux-j7rs:/home/sangola/Desktop/myFolder>  file    *
myCalculation.ods:        OpenDocument Spreadsheet
myword.odt:               OpenDocument Text
new file:                 empty
Folder1:                  directory

Rps@linux-j7rs:/home/sangola/Desktop/myFolder>  file    myCalculation.ods
myCalculation.ods:        OpenDocument Spreadsheet
```

6) ls command: (List)

- This command <u>is used to list out directory contents i.e. it display subdirectories, files contained in the pwd in alphabetical order</u>.

Syntax: ls OPTION FileOrDirectory

Here, at OPTION we can use:

–a : This option lists <u>all files in the current directory including hidden files</u>.

Note: To hide folder or file just rename file and append dot (.) at starting of file or folder.

–r : This option lists all files (all non-hidden) in reverse alphabetical order in the current directory.

–F : This option lists all files <u>by file type, here extra character is placed at the end of file</u>. For example forward slash(/) represents a directory & '@' represents a linked file that shows at the end of file.

–i : This option lists files with inode number (an inode number represent the location of a file on a Volume)

–l : This option lists all the files in the current working directory with details such as permission of file, owner of file, group of file, size of file, date and time of file, file name.

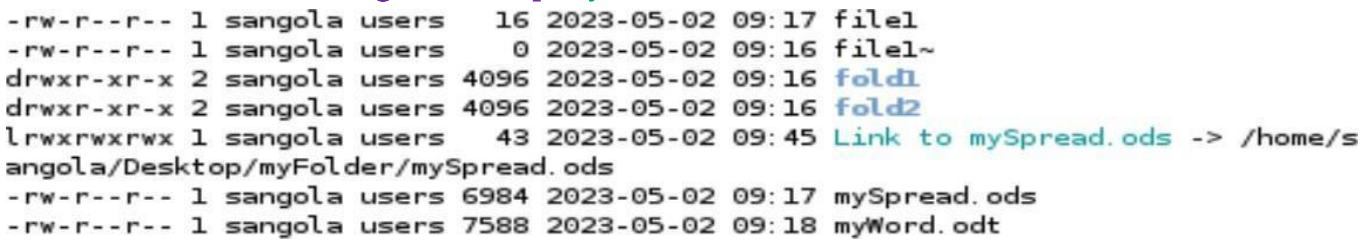
–t : This option lists the files by the last time <u>they were be changed</u>, here most recent files are listed first.

–u: This option lists the files by the last time <u>they were accessed,</u> most recent files are listed first.

Note: ls with no option and no file/Directory will display all files and directories in pwd.

Examples:

```
Rps@linux-j7rs:/home/sangola/Desktop/MyFolder> ls
file1     fold1     fold2     mySpread.ods      myWord.odt

Rps@linux-j7rs:/home/sangola/Desktop/MyFolder> ls    -a
file1     fold1     fold2     .hidenFold     mySpread.ods      myWord.odt
```

```
Rps@linux-j7rs:/home/sangola/Desktop/MyFolder>        -r
lsmyWord.odt          mySpread.ods        fold2       fold1        file1
Rps@linux-j7rs:/home/sangola/Desktop/MyFolder>ls   -F
file1     fold1/     fold2/     Link to mySpread.ods@      mySpread.ods       myWord.odt

Rps@linux-j7rs:/home/sangola/Desktop/MyFolder>ls   -i
181022   file1                181134  fold1            181031  fold2        181053  Link to mySpread.ods
181014   mySpread.ods         181015  myWord.odt

Rps@linux-j7rs:/home/sangola/Desktop/MyFolder>ls   -l
```


```
-rw-r--r-- 1 sangola users   16 2023-05-02 09:17 file1
-rw-r--r-- 1 sangola users    0 2023-05-02 09:16 file1~
drwxr-xr-x 2 sangola users 4096 2023-05-02 09:16 fold1
drwxr-xr-x 2 sangola users 4096 2023-05-02 09:16 fold2
lrwxrwxrwx 1 sangola users   43 2023-05-02 09:45 Link to mySpread.ods -> /home/s
angola/Desktop/myFolder/mySpread.ods
-rw-r--r-- 1 sangola users 6984 2023-05-02 09:17 mySpread.ods
-rw-r--r-- 1 sangola users 7588 2023-05-02 09:18 myWord.odt
Rps@linux-j7rs:/home/sangola/Desktop/MyFolder>ls   -l        myWord.odt
-rw-r—r--   1   sangola   users  7585    2023-05-0     09.20        myWord.odt

Rps@linux-j7rs:/home/sangola/Desktop/MyFolder>ls  -t
myWord.odt      Link to mySpread.ods          mySpread.ods          fold2        fold1          file1

Rps@linux-j7rs:/home/sangola/Desktop/MyFolder>ls   -u
Link to mySpread.ods       myWord.odt     mySpread.ods          fold1        fold2          file1
```

7) cat command:

- This cat command is multipurpose command, used basically for:
  - Create new file.
  - View/Read contents of file.
  - Copy contents of one file into another file.
  - Append contents into existing file.

Let's do these task using cat command.

1) Creating file using cat command:

cat command is used to create new file using following syntax:

Syntax: **cat > FileName**

Here, FileName is name of file that we are going to create.

Note: File is created in write mode.

Note: At file creation, mention extension of file also. If we do not mention extension then by default it will be considered as ASCII text file.

Example:

```
Rps@linux-j7rs:/home/sangola/Desktop/MyFolder>  cat    > myFile
Learning LINUX shell commands..
It's nice to learn..!
Note: Press Ctrl+D to exit.
```

2) Read or View content of file:

cat command is also used to view/read content of file using following syntax:

Syntax: **cat OPTION FileName**

Here, at OPTION we can use:

-n: This option displays contents in the form of line numbers

-E: This option display $ symbol at end of each line.

Note: We can also use cat command without option. Then it will display all content of file as it is.

Examples:

```
Rps@linux-j7rs:/home/sangola/Desktop/MyFolder>  cat   -n   myFile
1     Learning LINUX shell commands..
2     It's nice to learn..!

Rps@linux-j7rs:/home/sangola/Desktop/MyFolder>          -E   myFile
 catLearning LINUX shell commands..$
 It's nice to learn..!$

Rps@linux-j7rs:/home/sangola/Desktop/MyFolder>          myFile
 catLearning LINUX shell commands..
 It's nice to learn..!
```

3) Copy contents of one file into another file:

We can also copy the content of one file into another using cat command as follow-

Syntax: **cat** **oldFile > newFile**

Here, oldFile is source file and newFile is destination file.

Examples:

```
Rps@linux-j7rs:/home/sangola/Desktop/MyFolder>  cat   myFile >   newFile

See content of copied data as:
Rps@linux-j7rs:/home/sangola/Desktop/MyFolder> cat newFile
 Learning LINUX shell commands..
 It's nice to learn..!
```

4) Appending extra content into existing file:

We can append extra data into existing file using cat command as follow-

Syntax: **cat** >> **FileName**

```
Rps@linux-j7rs:/home/sangola/Desktop/MyFolder> cat >> newFile
This is extra data to append.
Note: Press Ctrl+D to exit.
See data of newFile as:
Rps@linux-j7rs:/home/sangola/Desktop/MyFolder> cat newFile
 Learning LINUX shell commands..
 It's nice to learn..!
 This is extra data to append.
```

8) more command:

- The more command displays the contents of a file one page at a time. To view the next page of the file, we have to **press the Space bar**. To quit viewing the file, we have to press the **q key**.
- The more command is simple and fast, but it doesn't allow you to scroll back up to view previously displayed text.
- That is more command allows only forward movement in the file.
- At the bottom of the screen, it also shows percentage of the file that viewed or displayed.

Syntax: **more** **OPTION** **FileName**

Here, at OPTION we can use:

-num specify **integer value** that denotes the number of lines per page to display.

Example:

```
Rps@linux-j7rs:/home/sangola/Desktop/MyFolder>              -3    BioData
moreMy Name is Rahul
I live in Pune.
I have 2 Brothers
--More--(30%)
Note: Press spacebar key to view next content of file.
      Press q key to exit view of the file.
```

9) less command:

- The functionality of less command is same as more command i.e. it displays the contents of a file one page at a time but, less command allows **forward and backward** movement in the file. (scroll up and down through the contents of a file)
- To view the next page of the file, we have to **press the Space bar**. To quit viewing the file, we have to press the **q key**.
- At the bottom of the screen, it also shows file name and number of lines that were viewed or displayed.

Syntax: **less FileName**

Example:

```
Rps@linux-j7rs:/home/sangola/Desktop/MyFolder> less    BioData
My Name is Rahul
I live in Pune.
I have 2 Brothers
BioData   lines  1-20/95     30%
Note: Press spacebar key to view next content of file
      Press q key to exit view of the file
```

Note: To notice the difference between more and less command, just scroll mouse after executing both commands. Then you will get difference.

II) File and Directory Operations Commands:

- The commands which are used to perform several operations such as copying, printing, moving, removing, finding etc. file or directory are called 'File and directory operations commands'.
- Linux provides several file and directory operation commands which are discussed below-

1) find command:

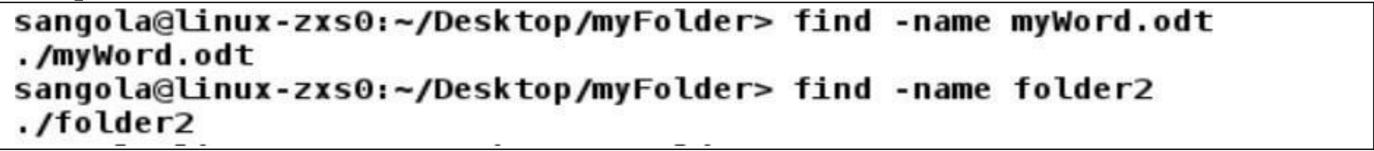
- The find command in Linux is a powerful utility that allows you to search for files and directories in a specified location on your system.
- We can search file based on various criteria, such as file name, group name, owner name, type, and permission.
- We can use find command by following manner.

1) Find file by file name:

Syntax: **find -name FileName**

Here, -name used to search file by name.

Examples:



```
sangola@linux-zxs0:~/Desktop/myFolder> find -name myWord.odt
./myWord.odt
sangola@linux-zxs0:~/Desktop/myFolder> find -name folder2
./folder2
```

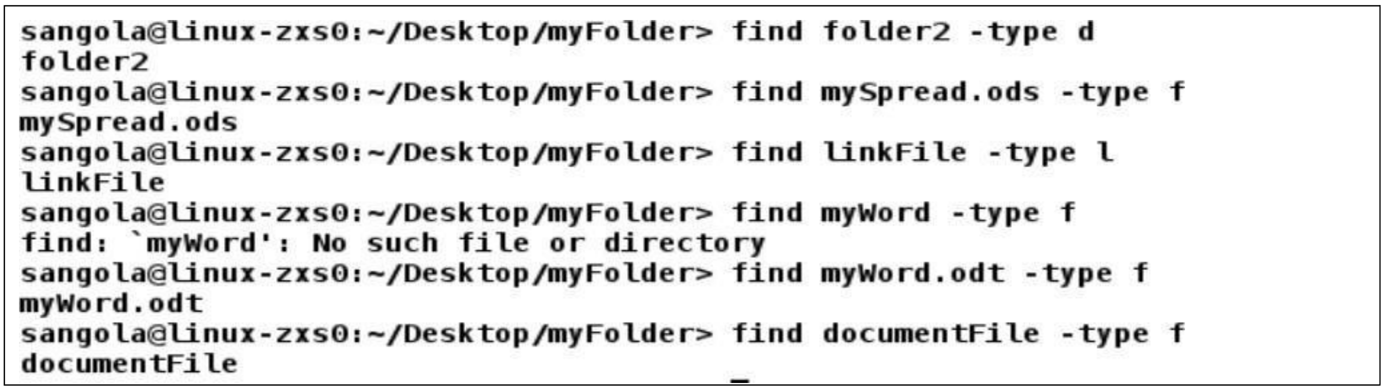
2) Find file by type:

Syntax: **find FileName -type FileTypeChar**

Here, at file type character we use character which is shown in following table:

File Type character	File Type
b	Block device
c	Character device
d	Directory
f	File
l	Symbolic link

Examples:

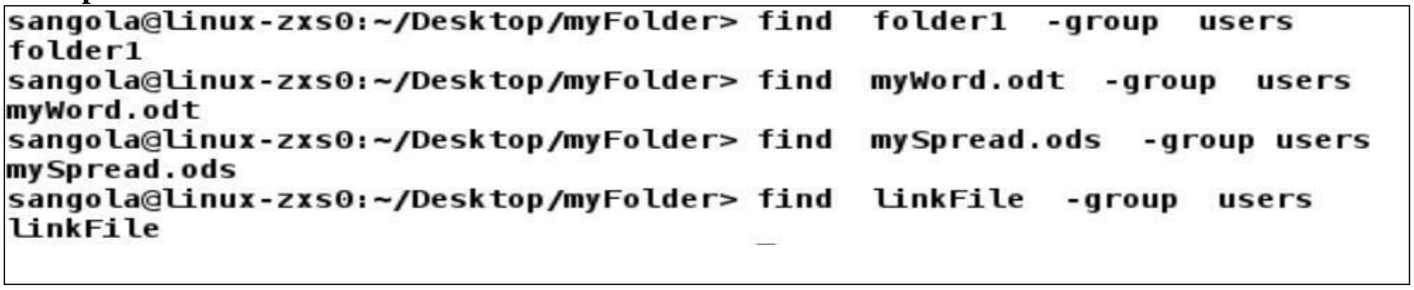


```
sangola@linux-zxs0:~/Desktop/myFolder> find folder2 -type d
folder2
sangola@linux-zxs0:~/Desktop/myFolder> find mySpread.ods -type f
mySpread.ods
sangola@linux-zxs0:~/Desktop/myFolder> find linkFile -type l
linkFile
sangola@linux-zxs0:~/Desktop/myFolder> find myWord -type f
find: `myWord': No such file or directory
sangola@linux-zxs0:~/Desktop/myFolder> find myWord.odt -type f
myWord.odt
sangola@linux-zxs0:~/Desktop/myFolder> find documentFile -type f
documentFile
```

3) Find file by group name:

Syntax: **find FileName -group groupName**

Example:

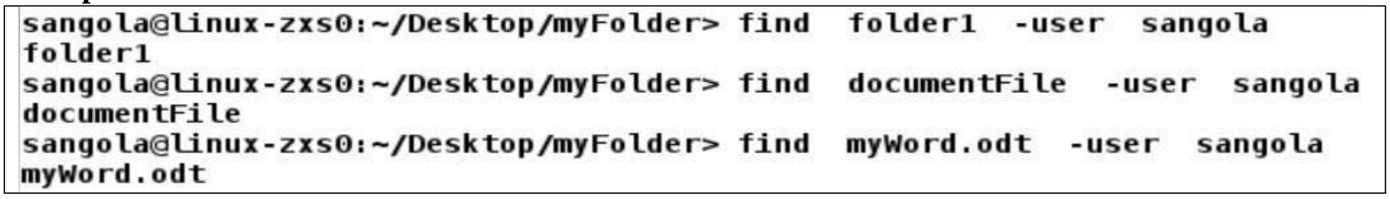


```
sangola@linux-zxs0:~/Desktop/myFolder> find  folder1  -group  users
folder1
sangola@linux-zxs0:~/Desktop/myFolder> find  myWord.odt  -group  users
myWord.odt
sangola@linux-zxs0:~/Desktop/myFolder> find  mySpread.ods  -group users
mySpread.ods
sangola@linux-zxs0:~/Desktop/myFolder> find  linkFile  -group  users
linkFile
```

4) Find file by user name (owner):

Syntax: **find FileName -user userName**

Examples:

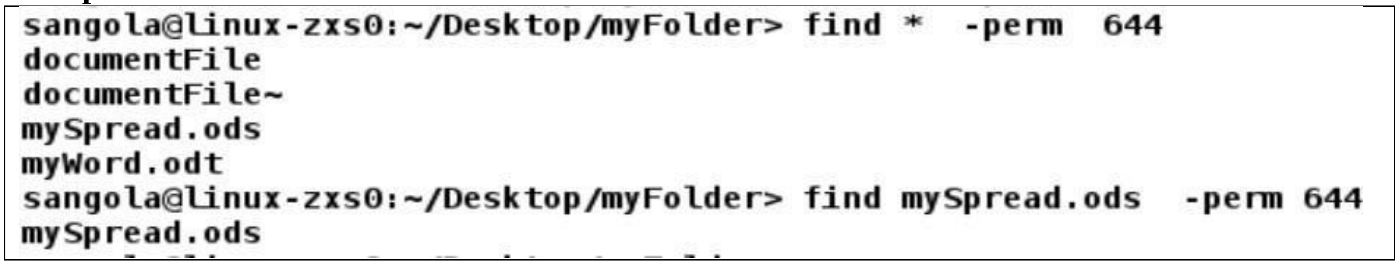


```
sangola@linux-zxs0:~/Desktop/myFolder> find  folder1  -user  sangola
folder1
sangola@linux-zxs0:~/Desktop/myFolder> find  documentFile  -user  sangola
documentFile
sangola@linux-zxs0:~/Desktop/myFolder> find  myWord.odt  -user  sangola
myWord.odt
```

5) Find file by permission:

Syntax: **find FileName -perm permOctalVal**

Examples:



```
sangola@linux-zxs0:~/Desktop/myFolder> find *  -perm  644
documentFile
documentFile~
mySpread.ods
myWord.odt
sangola@linux-zxs0:~/Desktop/myFolder> find mySpread.ods  -perm 644
mySpread.ods
```

Note: In above example, we use * that indicates all files

2) cp command :

➢ This command is used to make copy of files or directories.

Syntax:

cp OPTION srcFile destFile

Here, at OPTION we can use:

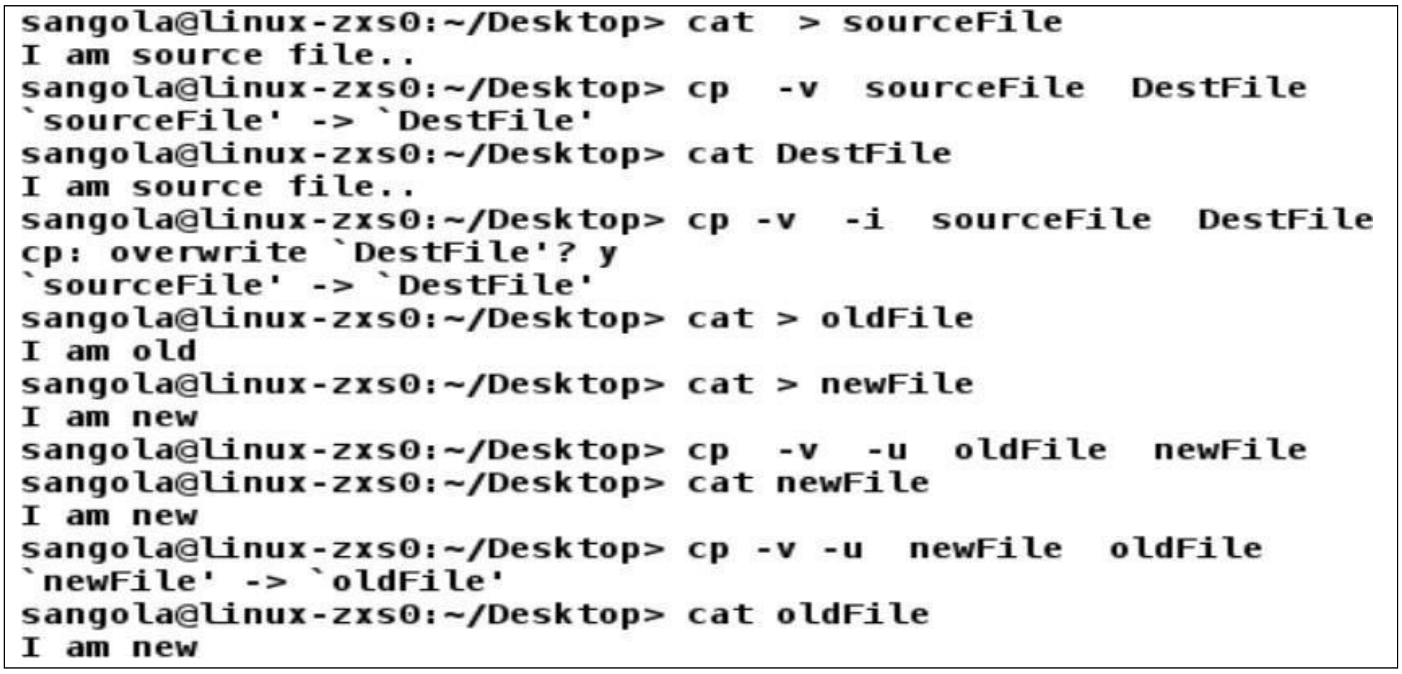
-i (Interactive prompt) ask for confirmation **before overwrite**.

-f (force) force to copy if an existing **destination file cannot be opened**, remove it and try again.

-u copy only when the **source file is newer than the destination file**.

-v (verbose) Display Message about what being done about copy.

Examples:



```
sangola@linux-zxs0:~/Desktop> cat  > sourceFile
I am source file..
sangola@linux-zxs0:~/Desktop> cp  -v  sourceFile  DestFile
`sourceFile' -> `DestFile'
sangola@linux-zxs0:~/Desktop> cat DestFile
I am source file..
sangola@linux-zxs0:~/Desktop> cp -v  -i  sourceFile  DestFile
cp: overwrite `DestFile'? y
`sourceFile' -> `DestFile'
sangola@linux-zxs0:~/Desktop> cat > oldFile
I am old
sangola@linux-zxs0:~/Desktop> cat > newFile
I am new
sangola@linux-zxs0:~/Desktop> cp  -v  -u  oldFile  newFile
sangola@linux-zxs0:~/Desktop> cat newFile
I am new
sangola@linux-zxs0:~/Desktop> cp -v -u  newFile  oldFile
`newFile' -> `oldFile'
sangola@linux-zxs0:~/Desktop> cat oldFile
I am new
```

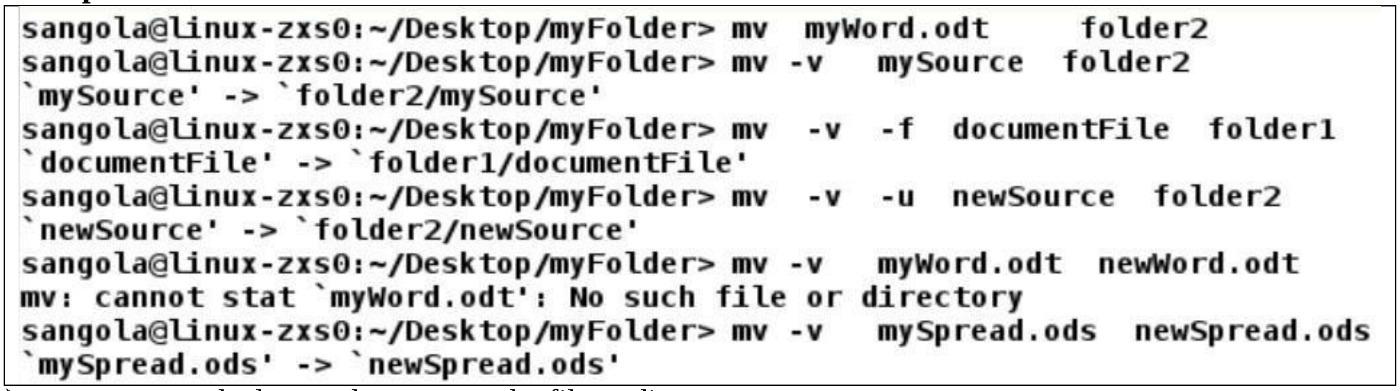
3) mv command:

- This command is used to move files and directories form one location to another location.

Syntax:

mv OPTION SourceFileOrFolder DestFolder

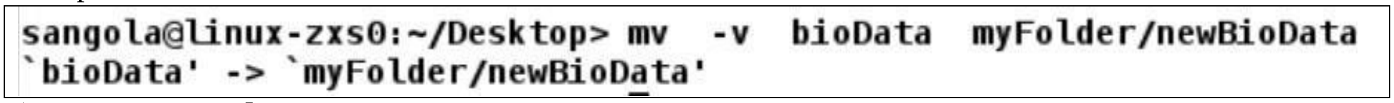
Here, at OPTION we can use:
-f : (force move) by overwriting destination file without prompt.
-i : (interactive prompt) **before overwrite**.
-u : (update) - move **when source is newer than destination.**
-v : (verbose) - print alert message regarding to move files
Examples:



```
sangola@linux-zxs0:~/Desktop/myFolder> mv  myWord.odt    folder2
sangola@linux-zxs0:~/Desktop/myFolder> mv -v   mySource  folder2
`mySource' -> `folder2/mySource'
sangola@linux-zxs0:~/Desktop/myFolder> mv  -v  -f  documentFile  folder1
`documentFile' -> `folder1/documentFile'
sangola@linux-zxs0:~/Desktop/myFolder> mv  -v  -u  newSource  folder2
`newSource' -> `folder2/newSource'
sangola@linux-zxs0:~/Desktop/myFolder> mv -v   myWord.odt  newWord.odt
mv: cannot stat `myWord.odt': No such file or directory
sangola@linux-zxs0:~/Desktop/myFolder> mv -v   mySpread.ods  newSpread.ods
`mySpread.ods' -> `newSpread.ods'
```

- mv command, also used to rename the file or directory.
- We can rename file or folder using mv command as-

Syntax: mv srcFileORFolder destFolder/newName
In above syntax, srcFileORFolder moved to destFolder with renamed file 'newName'
Example:



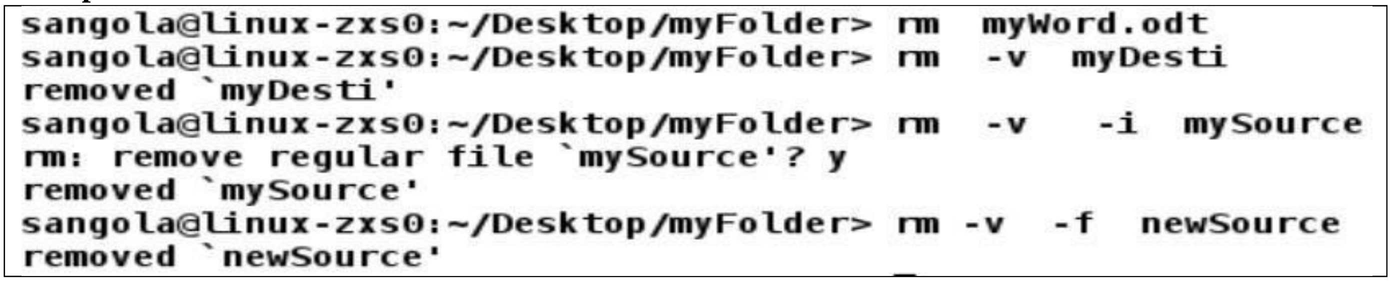
```
sangola@linux-zxs0:~/Desktop> mv  -v  bioData  myFolder/newBioData
`bioData' -> `myFolder/newBioData'
```

4) rm command:

- This command is used to remove/delete file(s) only.
- Note that, it does not remove directory.

Syntax: rm OPTION FileName
Here, at OPTION we can use:
-f: (force) it forces to remove files, **it never prompt**.

–i: prompt before every removal.
–v: print message regarding to remove command.
Examples:



```
sangola@linux-zxs0:~/Desktop/myFolder> rm   myWord.odt
sangola@linux-zxs0:~/Desktop/myFolder> rm   -v   myDesti
removed `myDesti'
sangola@linux-zxs0:~/Desktop/myFolder> rm   -v    -i   mySource
rm: remove regular file `mySource'? y
removed `mySource'
sangola@linux-zxs0:~/Desktop/myFolder> rm -v   -f   newSource
removed `newSource'
```

5) ln Command: (Link)

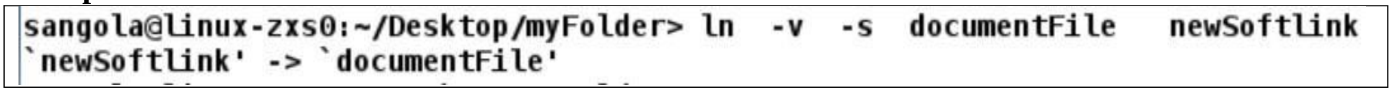
- This command is used to make links between files i.e. it creates short cuts for existing file.
- Basically, links are reference to a file using different filenames to access it.
- The assigned names are often referred as links.
- The links are of two types: **1) Symbolic (Soft) Link** **2) Hard Link**
- Let's see these links in details:

1) Symbolic link: **(soft link)**

- To create symbolic Link –s option is used with ln command which is given in following syntax:

Syntax: **ln –s sourceFileName linkFileName**

Example:



```
sangola@linux-zxs0:~/Desktop/myFolder> ln  -v  -s  documentFile   newSoftlink
`newSoftlink' -> `documentFile'
```

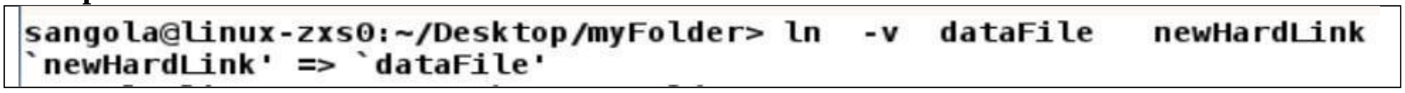
Note: When we remove or delete original file then NO REFERENCE to symbolic link file will EXIST because both files are holds same path/location

2) Hard Link:

- To create Hard Link ln command is run **without –s** option.

Syntax: **ln sourceFileName linkFileName**

Example:



```
sangola@linux-zxs0:~/Desktop/myFolder> ln   -v   dataFile    newHardLink
`newHardLink' => `dataFile'
```

Note: When we remove original file then REFERENCE to HARD link file will EXIST .i.e. sourceFile data is exist within hardLinkFile.

Printing the Files: (File printing commands)

- Linux OS also have commands which are used to print the files. They are explained as follow-

1) lpr command:

- This command is used to print file(s) on printer connected to system.

Syntax: **lpr OPTION fileName1 fileName2 fileName3**

Here, at OPTION we can use:

-num: Sets the number of copies to print from 1 to 100

–r: The named print file will be deleted after printing.

Examples:

```
Rps@linux-j7rs:/home/sangola/Desktop/MyFolder> lpr   -5   BioData
```
Note: The above command prints five copies of BioData file
```
Rps@linux-j7rs:/home/sangola/Desktop/MyFolder> lpr   BioData    myWord.odt
```
Note: The above command prints all pages of BioData and myWOrd.odt files
```
Rps@linux-j7rs:/home/sangola/Desktop/MyFolder> lpr   -r   BioData
```
Note: The above command prints all pages of BioData and then it will deleted.

2) lpq command: (Line Printer Queue)

- This command is used to show printer queue status.
- That is it list out the pending printing jobs on Printer in the queue.

Syntax: **lpq OPTION**

Here, at OPTION we can use:

-a: It reports list of pending jobs on ALL PRINTERS connected to system.

Example:

```
Rps@linux-j7rs:/home/sangola/Desktop/MyFolder> lpq
        Rank    Owner    Job      Files
        active  root     483      BioData.odt
        2nd     sangola  484      /tmp/myfile.txt
        3rd     Rps      485      reports.pdf
```

3) lprm command:

- This command is used to remove print request(s) from the Print Queue.
- It cancels print jobs that have been queued for Printing.

Syntax: **lprm JobID1 JobID2 JobID3**

Example:

```
Rps@linux-j7rs:/home/sangola/Desktop/MyFolder> lprm  484  485
```

Note: Above command will remove the print requests of sangola and Rps owner from the print queue.

III) Filter Commands:

- The role of filter commands in Linux is to manipulate and process data in a variety of ways, such as searching for specific patterns, extracting or deleting columns or fields, sorting data, and performing complex text processing tasks.
- Filter commands are often used in combination with other Linux commands to perform more advanced tasks.
- For example, grep can be used to search for a specific pattern in a file and then sed or awk can be used to modify the lines that match the pattern.
- Let's see several filter commands in details:

1) head Command:

- This command is used to display first N number of lines from a file.
- By default it prints first 10 number of lines of each given file.

Syntax: **head OPTION fileName**

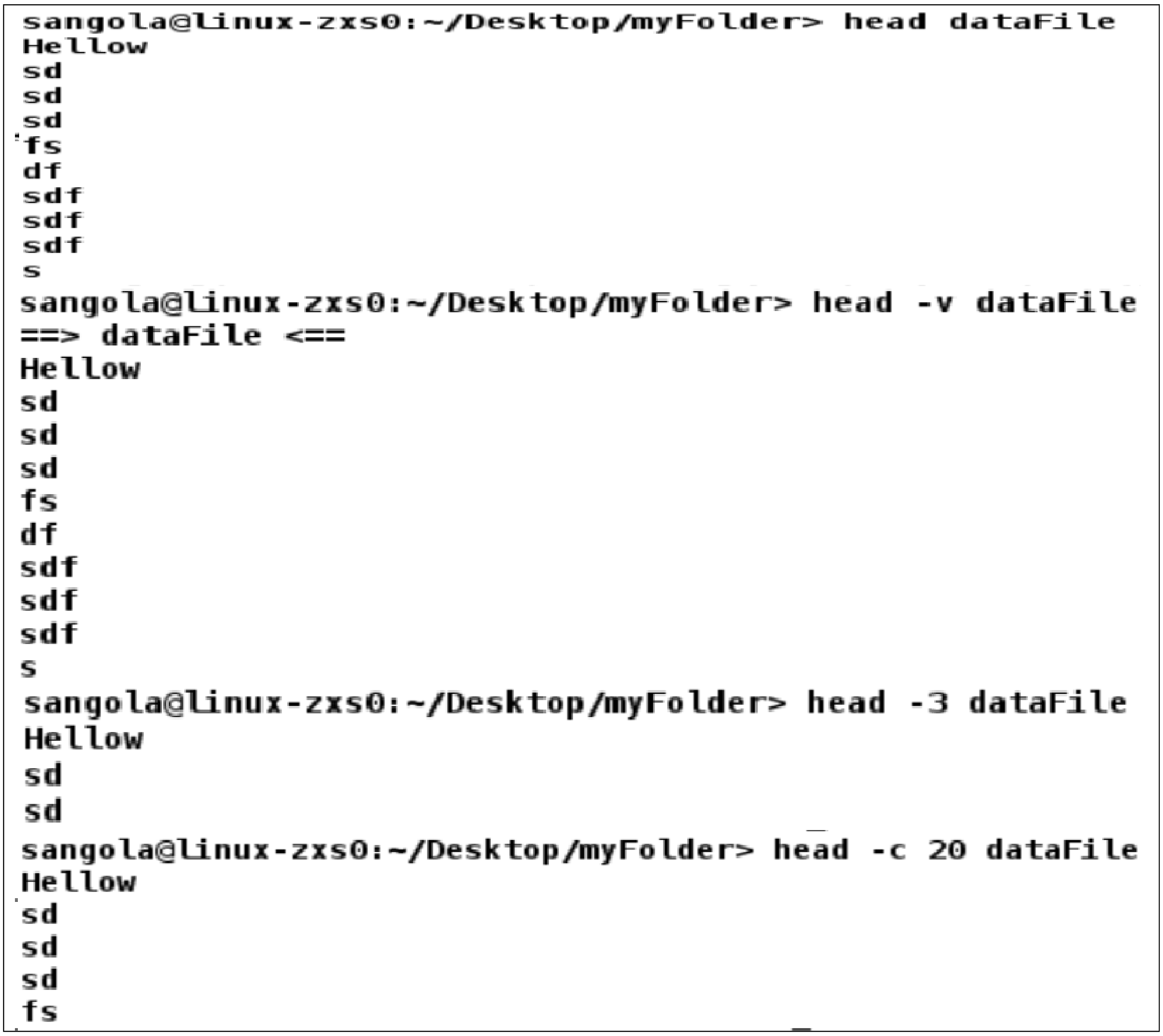
Here, at OPTION we can use:

-n: The n Specifies number of lines to be displayed.

-c: Specifies n number of bytes to be displayed.

-v: (verbose) that prints header information of file.

Examples: Consider we have **dataFile** file. Then we apply **head command** as follow-



```
sangola@linux-zxs0:~/Desktop/myFolder> head dataFile
Hellow
sd
sd
sd
fs
df
sdf
sdf
sdf
s
sangola@linux-zxs0:~/Desktop/myFolder> head -v dataFile
==> dataFile <==
Hellow
sd
sd
sd
fs
df
sdf
sdf
sdf
s
sangola@linux-zxs0:~/Desktop/myFolder> head -3 dataFile
Hellow
sd
sd
sangola@linux-zxs0:~/Desktop/myFolder> head -c 20 dataFile
Hellow
sd
sd
sd
fs
```

2) tail command-

- This command is used to <u>display LAST 'N' number</u> of data lines from file.
- By default <u>it prints LAST 10 numbers of lines</u> of each given file.

Syntax: **tail** **OPTION** **fileName**

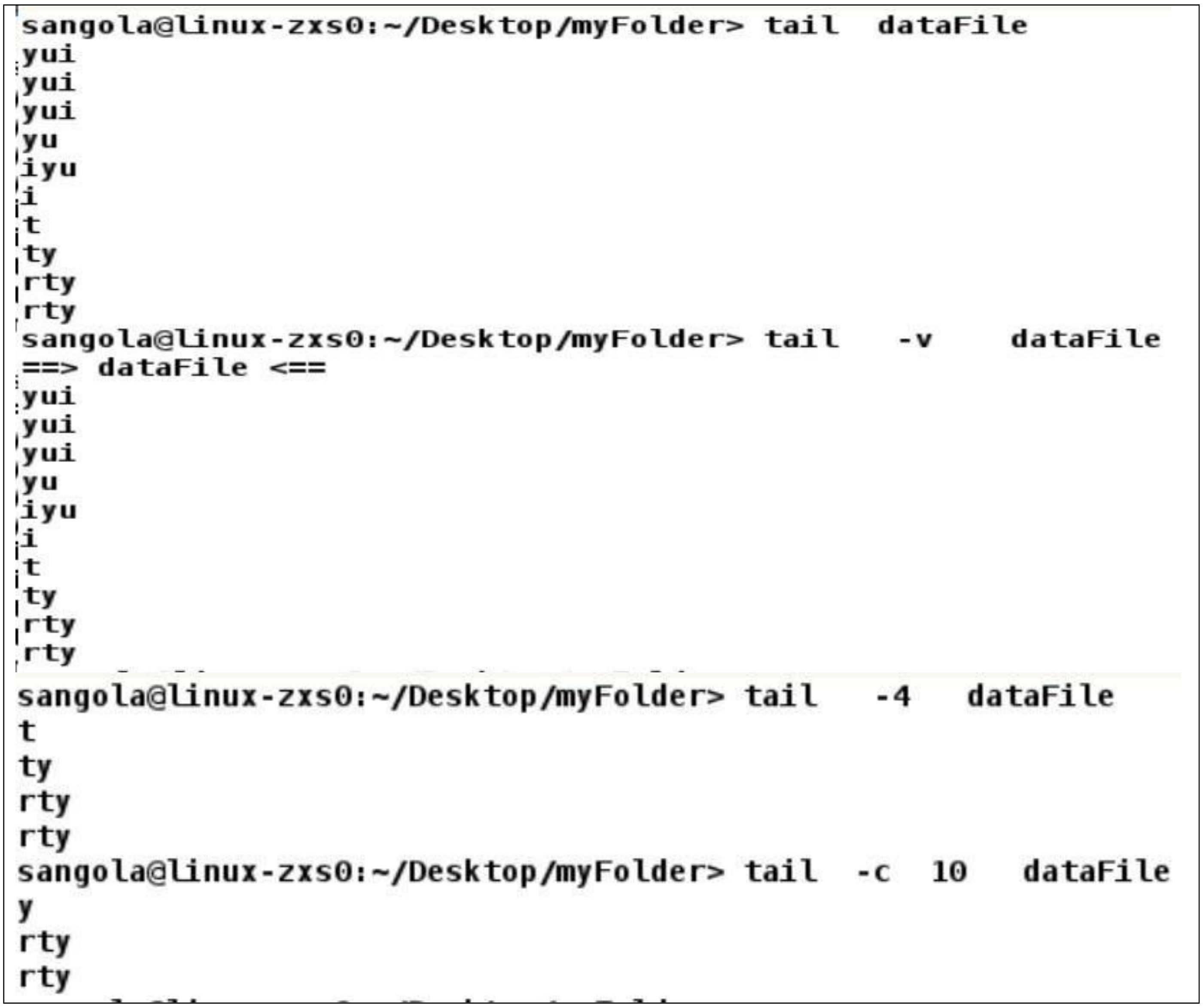
Here, at OPTION we can use:

-n: The n Specifies number of lines to be displayed.

-c: Specifies n number of bytes to be displayed.

-v: (verbose) that prints header information of file.

Examples: Consider we have **dataFile** file. Then we apply **tail command** as follow-



```
sangola@linux-zxs0:~/Desktop/myFolder> tail  dataFile
yui
yui
yui
yu
iyu
i
t
ty
rty
rty
sangola@linux-zxs0:~/Desktop/myFolder> tail   -v    dataFile
==> dataFile <==
yui
yui
yui
yu
iyu
i
t
ty
rty
rty
sangola@linux-zxs0:~/Desktop/myFolder> tail   -4   dataFile
t
ty
rty
rty
sangola@linux-zxs0:~/Desktop/myFolder> tail  -c  10   dataFile
y
rty
rty
```

3) pr command:

- ➢ The pr command <u>prepares a file for printing</u> by adding suitable header, footer, page number and formatted text for file.
- ➢ The header shows the date and time of last modification of the file along with the filename and page numbers.

Syntax: pr   OPTION   FileName

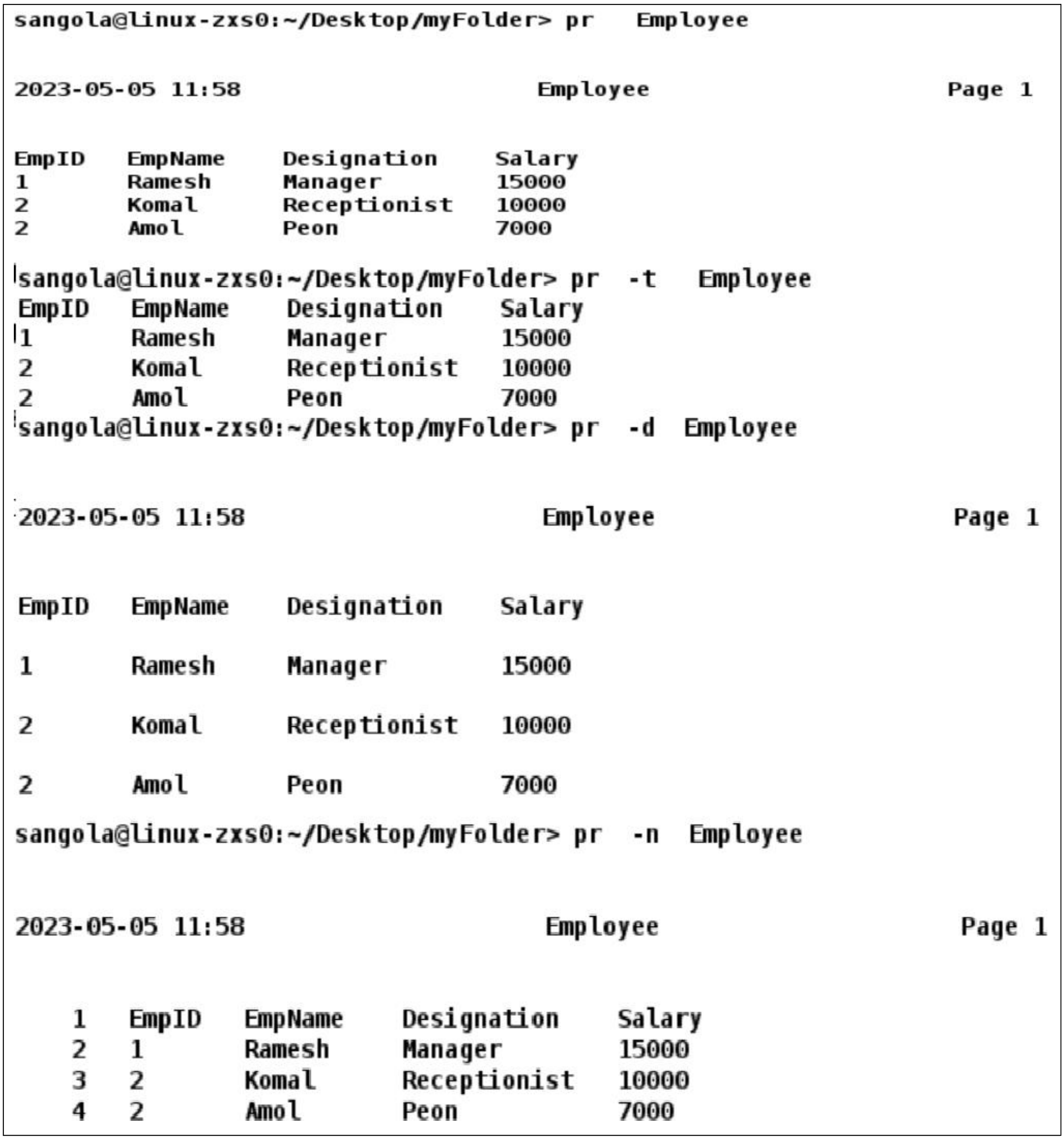
Here, at OPTION we can use:

-t: Omit the header and footer.

-d: Double spaces between each line.

-n: adding Numbers to lines.

Examples: Consider we have **Employee** file. Then we apply pr command as follow-



```
sangola@linux-zxs0:~/Desktop/myFolder> pr   Employee


2023-05-05 11:58                          Employee                         Page 1


EmpID   EmpName    Designation    Salary
1       Ramesh     Manager        15000
2       Komal      Receptionist   10000
2       Amol       Peon           7000

sangola@linux-zxs0:~/Desktop/myFolder> pr  -t   Employee
EmpID   EmpName    Designation    Salary
1       Ramesh     Manager        15000
2       Komal      Receptionist   10000
2       Amol       Peon           7000
sangola@linux-zxs0:~/Desktop/myFolder> pr  -d  Employee


2023-05-05 11:58                          Employee                         Page 1


EmpID   EmpName    Designation    Salary

1       Ramesh     Manager        15000

2       Komal      Receptionist   10000

2       Amol       Peon           7000
sangola@linux-zxs0:~/Desktop/myFolder> pr  -n  Employee


2023-05-05 11:58                          Employee                         Page 1


    1   EmpID   EmpName    Designation    Salary
    2   1       Ramesh     Manager        15000
    3   2       Komal      Receptionist   10000
    4   2       Amol       Peon           7000
```

4) cut command:

- This command <u>cuts/pick up a given number of characters' from each line</u> of specified file.
- Useful when having large database file and we have to pick a specific selected fields or characters.

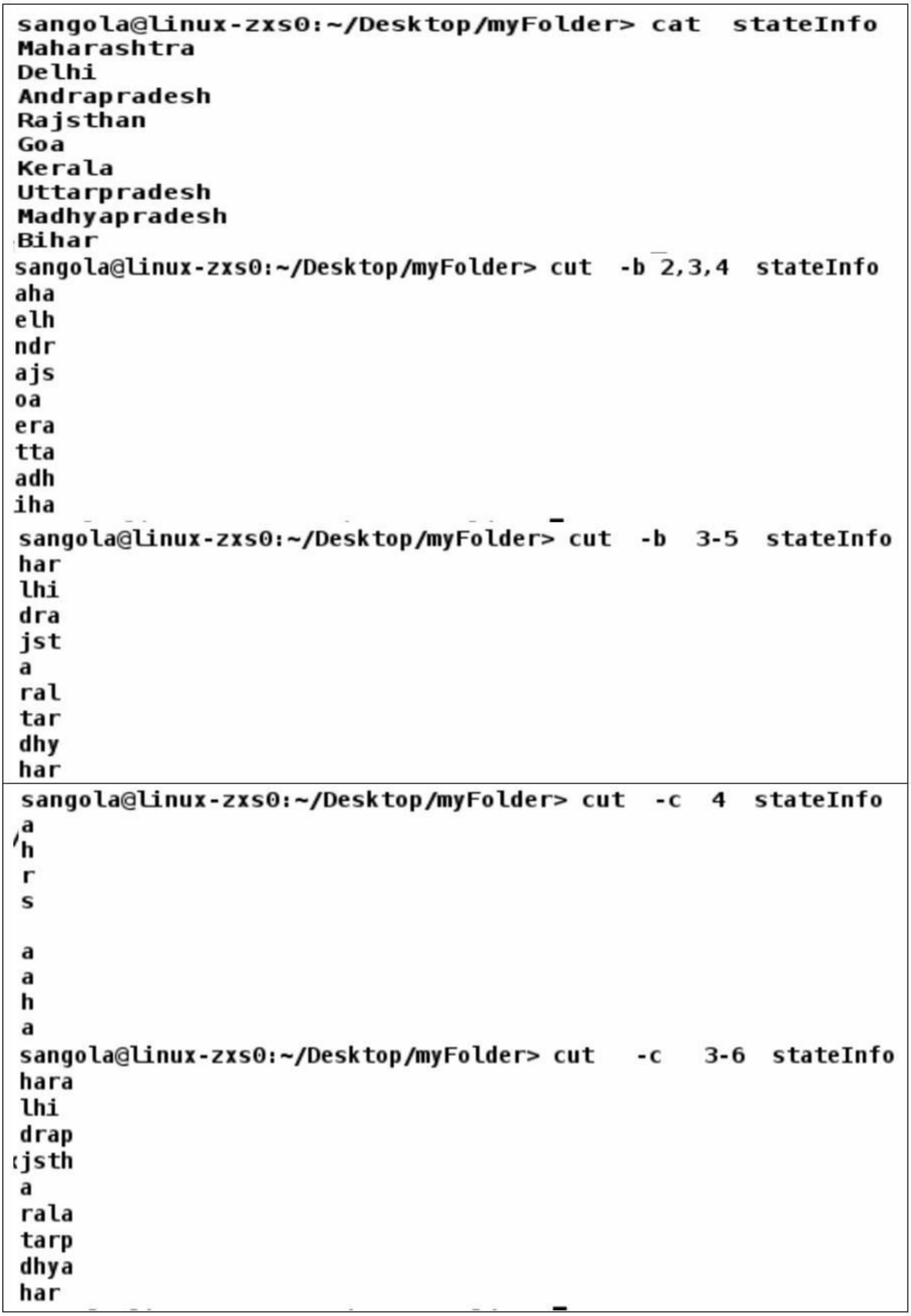
Syntax: **cut** **OPTION** **fileName**

Here, at OPTION we can use:

-b: bytes=LIST Select only the bytes from each line as specified in LIST. Here, LIST specifies a byte, a set of bytes, or a range of bytes.

-c: characters=LIST Select only the characters from each line as specified in LIST. Here, LIST specifies a character, a set of characters, or a range of characters.

Examples:



```
sangola@linux-zxs0:~/Desktop/myFolder> cat  stateInfo
Maharashtra
Delhi
Andrapradesh
Rajsthan
Goa
Kerala
Uttarpradesh
Madhyapradesh
Bihar
sangola@linux-zxs0:~/Desktop/myFolder> cut  -b 2,3,4  stateInfo
aha
elh
ndr
ajs
oa
era
tta
adh
iha
sangola@linux-zxs0:~/Desktop/myFolder> cut  -b  3-5  stateInfo
har
lhi
dra
jst
a
ral
tar
dhy
har
sangola@linux-zxs0:~/Desktop/myFolder> cut  -c  4  stateInfo
a
h
r
s

a
a
h
a
sangola@linux-zxs0:~/Desktop/myFolder> cut   -c   3-6  stateInfo
hara
lhi
drap
jsth
a
rala
tarp
dhya
har
```

5) paste command:

➢ This command is used to joins files together line by line.

Syntax: **paste OPTION fileName1 fileName2 fileName3**

Here, at OPTION we can use:

-d: It used to add delimiter symbol in between lines

–s: It joins all lines in a single line.

Examples: Let us consider we have three files **state**, **capital** and **number**. Then we use paste command as:

```
Rps@linux-j7rs:/home/sangola/Desktop/MyFolder>          state
catArunachal Pradesh
Assam
Andhra Pradesh
Bihar
Chhattisgrah
Rps@linux-j7rs:/home/sangola/Desktop/MyFolder>          capital
catItanagar
Dispur
Hyderabad
Patna
Raipur
Rps@linux-j7rs:/home/sangola/Desktop/MyFolder>          number
cat1
2
3
4
5
Rps@linux-j7rs:/home/sangola/Desktop/MyFolder>  paste  number   state   capital
1     Arunachal Pradesh      Itanagar
2     Assam                  Dispur
3     Andhra Pradesh         Hyderabad
4     Bihar                  Patna
5     Chhattisgrah           Raipur
Rps@linux-j7rs:/home/sangola/Desktop/MyFolder> paste -d “|”     number   state   capital
 1|Arunachal Pradesh|Itanagar
2|Assam|Dispur
3|Andhra Pradesh|Hyderabad
4|Bihar|Patna
5|Chhattisgrah|Raipur
```

6) sort command:

➢ This command is used to sort a file, it arranges the records in a particular order.

Syntax: **sort OPTION FileName**

Here, at OPTION we can use-

-k: To sort data of file based on a field number in a file. i.e. -k **2** (Here, 2 is field no)

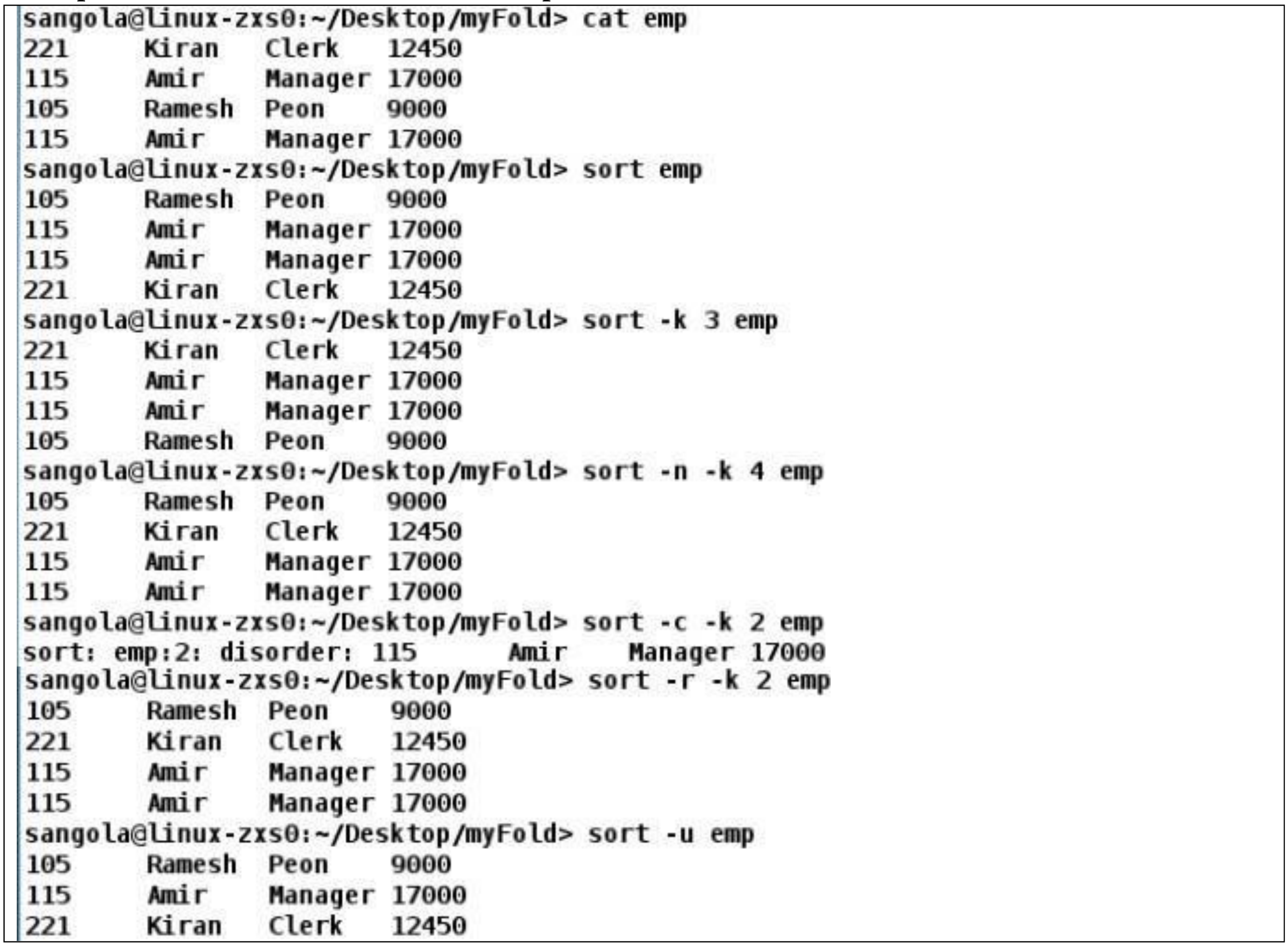
-r: It sort data to reverse sequence.

-n: To sort data of file numerically.

-c: Check data of file is sorted or not.

-u: Removes repeated lines**.**

Examples: Let us consider we have a file **emp** as



```
sangola@linux-zxs0:~/Desktop/myFold> cat emp
221     Kiran   Clerk   12450
115     Amir    Manager 17000
105     Ramesh  Peon    9000
115     Amir    Manager 17000
sangola@linux-zxs0:~/Desktop/myFold> sort emp
105     Ramesh  Peon    9000
115     Amir    Manager 17000
115     Amir    Manager 17000
221     Kiran   Clerk   12450
sangola@linux-zxs0:~/Desktop/myFold> sort -k 3 emp
221     Kiran   Clerk   12450
115     Amir    Manager 17000
115     Amir    Manager 17000
105     Ramesh  Peon    9000
sangola@linux-zxs0:~/Desktop/myFold> sort -n -k 4 emp
105     Ramesh  Peon    9000
221     Kiran   Clerk   12450
115     Amir    Manager 17000
115     Amir    Manager 17000
sangola@linux-zxs0:~/Desktop/myFold> sort -c -k 2 emp
sort: emp:2: disorder: 115      Amir    Manager 17000
sangola@linux-zxs0:~/Desktop/myFold> sort -r -k 2 emp
105     Ramesh  Peon    9000
221     Kiran   Clerk   12450
115     Amir    Manager 17000
115     Amir    Manager 17000
sangola@linux-zxs0:~/Desktop/myFold> sort -u emp
105     Ramesh  Peon    9000
115     Amir    Manager 17000
221     Kiran   Clerk   12450
```

7) uniq command:

This command is used for following purpose:

- The uniq command filters and removes repeated lines from file.
- When we concatenate or merge files, we will face the duplicate lines problem then uniq command reduces these duplicate lines problem.

Syntax: **uniq OPTION FileName**

Here, at OPTION we can use:

–u: It selects only Lines that are NOT repeated
–d: It selects only one copy of repeated lines.
–c: It displays the frequency of occurrence of all lines along with lines.

NOTE: uniq command requires a sorted file as input, so first we have to sort a file and then **pipe (|)** the process to uniq command. Which is shown below,

sort FileName | uniq OPTION

Examples: Let us consider we have **deptData** file as-

```
Rps@linux-j7rs:/home/sangola/Desktop/MyFolder>  cat
01      kalyan    accounts    6213
02      satish    office      4214
01      kalyan    accounts    6213
03      murthy    office      4214
03      murthy    office      4214
Rps@linux-j7rs:/home/sangola/Desktop/MyFolder>  sort  deptData |  uniq
01      kalyan    accounts    6213
02      satish    office      4214
```

```
03      murthy    office      4214
Rps@linux-j7rs:/home/sangola/Desktop/MyFolder>  sort   deptData |  uniq    -c
 2    01      kalyan     accounts    6213
 1    02      satish      office      4214
 2    03      murthy     office      4214
Rps@linux-j7rs:/home/sangola/Desktop/MyFolder>  sort  deptData |  uniq     -u
02      satish      office      4214
Rps@linux-j7rs:/home/sangola/Desktop/MyFolder>  sort   deptData |  uniq    -d
01      kalyan     accounts    6213
03      murthy    office      4214
```

8) grep command- (Global Regular Expression Print)

- The grep filter command searches given pattern of characters (i.e. regular expression) into file or directory, and displays all lines from file that contains the given pattern of characters.
- The pattern which will be searched in the file is referred as regular expression.

Syntax: **grep   OPTION   pattern   FileName**

Here, at OPTION we can use-

-c: This prints only a **count of the lines** that match a pattern.
-h: Display all lines that matches the pattern.
-i: Ignores, case for matching pattern.
-l: Displays list of a filenames only that matches the pattern.
-n: Display the matched lines and their line numbers.
-v: This prints out all the lines that do not matches the pattern
-o: Print only the matched pattern with each matching line.
–w: Print whole line that match whole words with each line of file.
Examples: Let us we have **myOS** file as follow:

```
Rps@linux-j7rs:/home/sangola/Desktop/MyFolder>  cat  myOS
unix is great operating system. unix is open source. unix is free.
learn operating system.
unixlinux which one you choose.
uNix is easy to learn. Unix is multi user os. Learnunix. unix is powerful
Rps@linux-j7rs:/home/sangola/Desktop/MyFolder>  grep  -c    unix  myOS
3
Rps@linux-j7rs:/home/sangola/Desktop/MyFolder>  grep  -h    unix myOS
unix is great operating system. unix is open source. unix is free.
unixlinux which one you choose.
uNix is easy to learn. Unix is multi user os. Learnunix. unix is powerful
Rps@linux-j7rs:/home/sangola/Desktop/MyFolder>  grep  -i    UNIX myOS
unix is great operating system. unix is open source. unix is free.
unixlinux which one you choose.
uNix is easy to learn. Unix is multi user os. Learnunix. unix is powerful
Rps@linux-j7rs:/home/sangola/Desktop/MyFolder>  grep  -l    unix myOS
myOS
Rps@linux-j7rs:/home/sangola/Desktop/MyFolder>  grep  -n    unix myOS
1:unix is great operating system. unix is open source. unix is free.
3:unixlinux which one you choose.
4:uNix is easy to learn. Unix is multi user os. Learnunix. unix is powerful
Rps@linux-j7rs:/home/sangola/Desktop/MyFolder>  grep  -v    unix myOS
learn operating system.
Rps@linux-j7rs:/home/sangola/Desktop/MyFolder>  grep  -o    learn myOS
learn
```

```
learn
Rps@linux-j7rs:/home/sangola/Desktop/MyFolder>  grep  -w  operating    myOS
unix is great operating system. unix is open source. unix is free.
learn operating system.
```

9) egrep command: (Extended Global Regular Expression Print)

- A command egrep (Extended Global Regular Expression Print) is used to search for text patterns in one or more files. egrep includes a few extra features that give it more flexibility and power than grep command.
- egrep uses "extended regular expressions," which is more sophisticated kind of regular expression, more complicated pattern matching, which includes the ability to specify repetition and match numerous options. This was not found in grep command and this is difference between grep and egrep.
- Searching for certain words or phrases in a big collection of text files, searching for lines that follow a given pattern, or extracting specific data from log files are all operations done easily by egrep command.

Syntax: **egrep** **OPTION** **pattern** **FileName**

Here, at OPTION we can use-

-c: This prints only a count of the lines that match a pattern.
-h: Display all lines that matches the pattern.
-i: Ignores, case for matching pattern.
-l: Displays list of a filenames only that matches the pattern.
-n: Display the matched lines and their line numbers.
-v: This prints out all the lines that do not matches the pattern
-o: Print only the matched pattern with each matching line.
–w: Print whole line that match whole words with each line of file.

Note: All options of grep command can be execute with egrep command which is shown in examples

Examples: Let us we have **myOS** file as follow:

```
Rps@linux-j7rs:/home/sangola/Desktop/MyFolder>  cat  myOS
unix is great operating system. unix is open source. unix is free.
learn operating system.
unixlinux which one you choose.
uNix is easy to learn. Unix is multi user os. Learnunix. unix is powerful
Rps@linux-j7rs:/home/sangola/Desktop/MyFolder>  egrep  -c    unix  myOS
3
Rps@linux-j7rs:/home/sangola/Desktop/MyFolder>  egrep  -h    unix  myOS
unix is great operating system. unix is open source. unix is free.
unixlinux which one you choose.
uNix is easy to learn. Unix is multi user os. Learnunix. unix is powerful
Rps@linux-j7rs:/home/sangola/Desktop/MyFolder>  egrep  -i    UNIX  myOS
unix is great operating system. unix is open source. unix is free.
unixlinux which one you choose.
uNix is easy to learn. Unix is multi user os. Learnunix. unix is powerful
Rps@linux-j7rs:/home/sangola/Desktop/MyFolder>  egrep  -l    unix  myOS
myOS
Rps@linux-j7rs:/home/sangola/Desktop/MyFolder>  egrep  -n    unix  myOS
1:unix is great operating system. unix is open source. unix is free.
3:unixlinux which one you choose.
4:uNix is easy to learn. Unix is multi user os. Learnunix. unix is powerful
Rps@linux-j7rs:/home/sangola/Desktop/MyFolder>  egrep  -v    unix  myOS
learn operating system.
Rps@linux-j7rs:/home/sangola/Desktop/MyFolder>  egrep  -o    learn  myOS
learn
learn
```

```
Rps@linux-j7rs:/home/sangola/Desktop/MyFolder>  egrep  -w  operating     myOS
unix is great operating system. unix is open source. unix is free.
learn operating system.
```

10) fgrep command: (Fixed Global Regular Expression Print)

- A command fgrep (Fixed Global Regular Expression Print) is used to search for text patterns in one or more files.
- Unlike the grep command, which is used for searching for regular expressions, fgrep command is better at searching for fixed strings.
- This command is useful when you need to search for strings which contain lots of fixed regular expression metacharacters, such as "^", "$", "*", ">", "@" etc.
- fgrep is important and faster than grep in searching for fixed strings, especially when searching for multiple strings at one time. This is one of the main benefits of using fgrep. Because of this, fgrep is an excellent option for tasks like surveying through a huge collection of text files for particular words.

- **Note:** fgrep command is used to search for a specified string and not pattern in file.
  - **Syntax:**   **fgrep**   **OPTION**   **pattern**   **FileName**
  - Here, at OPTION we can use-

-c: This prints only a count of the lines that match a pattern.
-h: Display all lines that matches the pattern.
-i: Ignores, case for matching pattern.
-l: Displays list of a filenames only that matches the pattern.
-n: Display the matched lines and their line numbers.
-v: This prints out all the lines that do not matches the pattern
-o: Print only the matched pattern with each matching line.
–w: Print whole line that match whole words with each line of file.

Note: All options of grep command can be execute with fgrep command which is shown in examples

- **Examples:** Let us we have **myOS** file as follow:

```
Rps@linux-j7rs:/home/sangola/Desktop/MyFolder>  cat  myOS
unix is great operating system. unix is open source. unix is free.
learn operating system.
unixlinux which one you choose.
uNix is easy to learn. Unix is multi user os. Learnunix. unix is powerful
Rps@linux-j7rs:/home/sangola/Desktop/MyFolder>  fgrep  -c    unix  myOS
3
Rps@linux-j7rs:/home/sangola/Desktop/MyFolder>  fgrep  -h    unix  myOS
unix is great operating system. unix is open source. unix is free.
unixlinux which one you choose.
uNix is easy to learn. Unix is multi user os. Learnunix. unix is powerful
Rps@linux-j7rs:/home/sangola/Desktop/MyFolder>  fgrep  -i    UNIX  myOS
unix is great operating system. unix is open source. unix is free.
unixlinux which one you choose.
uNix is easy to learn. Unix is multi user os. Learnunix. unix is powerful
Rps@linux-j7rs:/home/sangola/Desktop/MyFolder>  fgrep  -l    unix  myOS
myOS
Rps@linux-j7rs:/home/sangola/Desktop/MyFolder>  fgrep  -n    unix  myOS
1:unix is great operating system. unix is open source. unix is free.
3:unixlinux which one you choose.
4:uNix is easy to learn. Unix is multi user os. Learnunix. unix is powerful
Rps@linux-j7rs:/home/sangola/Desktop/MyFolder>  fgrep  -v    unix  myOS
learn operating system.
```

```
Rps@linux-j7rs:/home/sangola/Desktop/MyFolder>  fgrep  -o    learn myOS
learn
learn
Rps@linux-j7rs:/home/sangola/Desktop/MyFolder>  fgrep  -w  operating     myOS
unix is great operating system. unix is open source. unix is free.
learn operating system.
```

11) tr command:(Translate Characters)

- This command is used to <u>translating or deleting characters of a file</u>.
- It supports a range of operations including uppercase to lowercase, squeezing repeating characters, deleting specific characters and find and replace characters.

Syntax: **tr OPTION SET1 SET2**

SET1 is source character set.

SET2 is destination character set.

Here, at OPTION we can use:

–d: This option use to deleting characters.

–t: This option used to find and replace character.

–s: This option used to squeezing repeating character from a file.

Examples: Let us consider we have **myData** file as

```
Rps@linux-j7rs:/home/sangola/Desktop/MyFolder>  cat     myData
linux is open source operating system. Developed by Linus Torvalds.
Rps@linux-j7rs:/home/sangola/Desktop/MyFolder>  cat     myData | tr  “a-z”    “A-Z”
LINUX IS OPEN SOURCE OPERATING SYSTEM. DEVELOPED BY LINUX TORVALDS.
Rps@linux-j7rs:/home/sangola/Desktop/MyFolder>  cat     myData | tr  -d     “o”
linux is pen surce perating system. Develped by Linus Trvalds.
Rps@linux-j7rs:/home/sangola/Desktop/MyFolder>  cat     myData | tr  -t    “e” “E”
linux is opEn sourcE opErating systEm. DEvElopEd by Linus Torvalds.
Rps@linux-j7rs:/home/sangola/Desktop/MyFolder> echo “Welcome        Linux” | tr -s   “ ”
Welcome Linux
```

IV) Archive and File Compression Commands:

- If file <u>size is large and we have to send it over the network from one client to another client then it took much more time because of large file size</u>.
- Also, there is issue of network. If network speed is slow and we are going to send large file then it also take much time and this process becomes lengthy.
- To avoid such kind of problems, linux operating system have commands that <u>archive and compress the large file</u>.
- After achieve and compression of file, its size becomes small and also content of file remains same. This is advantage of file compression and archive. Now, we can easily send such achieve and compressed file over the network easily.
- Let’s see some file archive and compression commands:

1) tar command- (.tar file)

- The tar command allows you to <u>pack up and compress files into **an archive format** and also extract it back as original whenever necessary</u>.
- It provides <u>loss less data compression</u> (it has some image compression algorithms) so that we get back out exactly what you put in compressed file, also it can take up less memory space when archived.
- <u>Also, tar command are often used for data backup, archival of large files that may send over the network.</u>

Syntax: **tar OPTION FileName.tar File1.ext1 File2.ext2 File3.ext3**

Here, at OPTION we can use-

-c: Create an archive.
-x: Extract from an archive.
-t: Display files in archive.
-f: It specifies the file name of the archive.
-v: Provide more verbose output related with file archive and compression.

Examples: Consider we have two text files named **first.odt** and **second.odt** as:

```
Rps@linux-j7rs:/home/sangola>  cat     first.odt
Hello,
How are you…!

Rps@linux-j7rs:/home/sangola> cat      second.odt
Hi,
I am fine, What about you all…!
Rps@linux-j7rs:/home/sangola> tar      -cvf      backup.tar       *.odt
first.odt
second.odt
Note: Above command make backup.tar achieve file of first.odt and second.odt files into pwd.

Rps@linux-j7rs:/home/sangola> tar      -xvf      backup.tar
first.odt
second.odt
Note: Above command extracts backup.tar achieve file into pwd.
```

2) zip and unzip command:

- The zip command also allows you to pack up & compress files **into a zip format**.
- It provides loss less data compression (it has some image compression algorithms) so that we get back out exactly what you put in compressed file, also it can take up less space when zipped.
- Also, zip command are often used for data backup, archival of large files that may send over the network.
- zip command requires the first argument to be compressed filename, and the remaining arguments are files or directories to be compressed.

Syntax: zip compressedFilename.zip File1.ext1 File2.ext2 File3.ext3

unzip command:

- This command is used to extract files and directories from zip file into pwd.

Syntax: unzip compressedFilename.zip

Examples: Consider we have two text files named **webpage.html** and **biodata.odt** as:

```
Rps@linux-j7rs:/home/sangola>  cat     webpage.html
<html>
<body>
<h1> My home Page </h1>
</body>
</html>
Rps@linux-j7rs:/home/sangola> cat      biodata.odt
Name: Rahul Mahesh Deokar
DOB: 20-06-2000
Address: Pune
Occup: Software developer
Rps@linux-j7rs:/home/sangola>          zipfile.zip     webpage.html      biodata.odt
zip
adding: webpage.html (deflated 19%)
Note: Above command make new zipfile.zip file of webpage.html and biodata.odt files into pwd.
```

```
Rps@linux-j7rs:/home/sangola> unzip   zipfile.zip
Archive:     zipfile.zip
Inflating:   webpage.html
Extracting: biodata.odt
```
Note: Above command extracts zipfile.zip achieve file into pwd.

V) Other Commands:

1) cal Command

- This command is used to see the calendar of any specific month or a complete year.

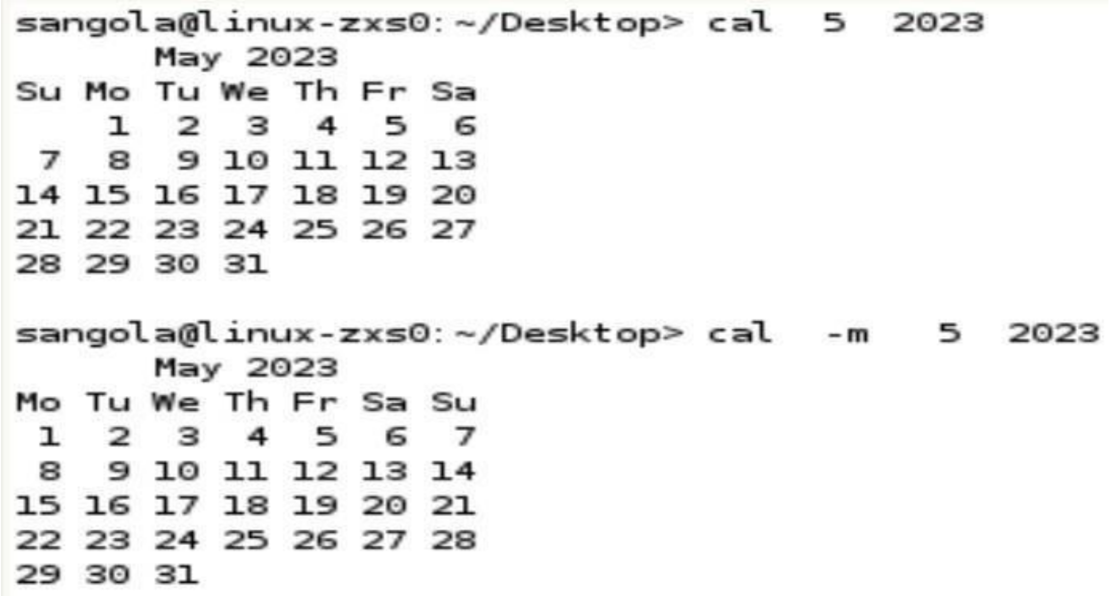
Syntax: cal OPITION Month Year

Here, at OPTION we can use-

–m: To set Monday as start of Week.

–j: To display Calendar in JULIAN Format.

Examples:



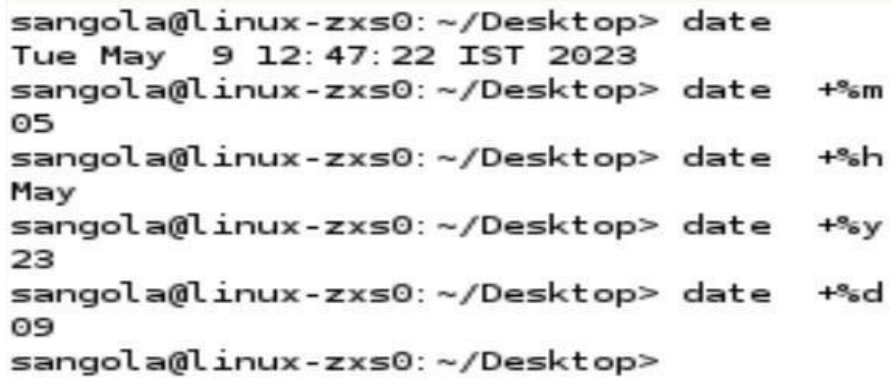
```
sangola@linux-zxs0:~/Desktop> cal   5   2023
      May 2023
Su Mo Tu We Th Fr Sa
    1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31

sangola@linux-zxs0:~/Desktop> cal   -m   5   2023
      May 2023
Mo Tu We Th Fr Sa Su
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31
```

2) date command:

- This command is used to display the current date.
- It shows the date and time to nearest second.

Syntax: date OPTION

Examples:

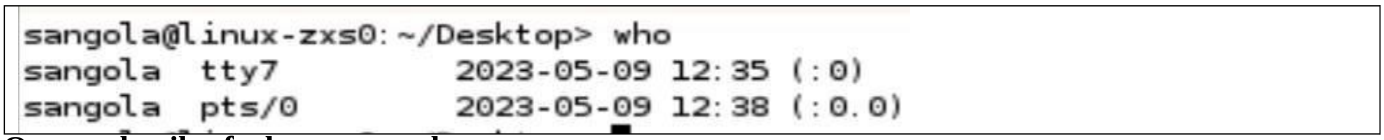


```
sangola@linux-zxs0:~/Desktop> date
Tue May  9 12:47:22 IST 2023
sangola@linux-zxs0:~/Desktop> date   +%m
05
sangola@linux-zxs0:~/Desktop> date   +%h
May
sangola@linux-zxs0:~/Desktop> date   +%y
23
sangola@linux-zxs0:~/Desktop> date   +%d
09
sangola@linux-zxs0:~/Desktop>
```

3) who command:

- who command is used to print information about user **who is currently logged in** to the system.
- **Note:** who command only see a real user who logged in (if we switch to user root and run who command still it shows real logged in user details).
- **Syntx: who**

Example:



```
sangola@linux-zxs0:~/Desktop> who
sangola  tty7         2023-05-09 12:35 (:0)
sangola  pts/0        2023-05-09 12:38 (:0.0)
```

Output details of who command:

1st column show the user name.

2nd column show how the user connected.

- tty means the user is connected directly to the computer (shows terminal ID)
- pts means the user is connected from remote.

3rd and 4th columns show the date and time

5th column show the IP Address where the users are connected.

Text Editor:

- ➢ A text editor is a software which is used to <u>create, edit, and modify plain text files</u>. It is a simple tool that allows you to type, edit, and format text.
- ➢ Unlike word processors, which are used to create and format documents, text editors are basically <u>used to write and edit code, scripts, and other text-based files</u>.
- ➢ Text editors are essential tools for programmers, developers, and anyone who works with code or text-based files. They provide a streamlined and lightweight interface that allows you to focus on writing and editing text without any distractions.
- ➢ Text editors also provide features such as syntax highlighting, auto-indentation, and code completion, which make it easier to write and edit code.

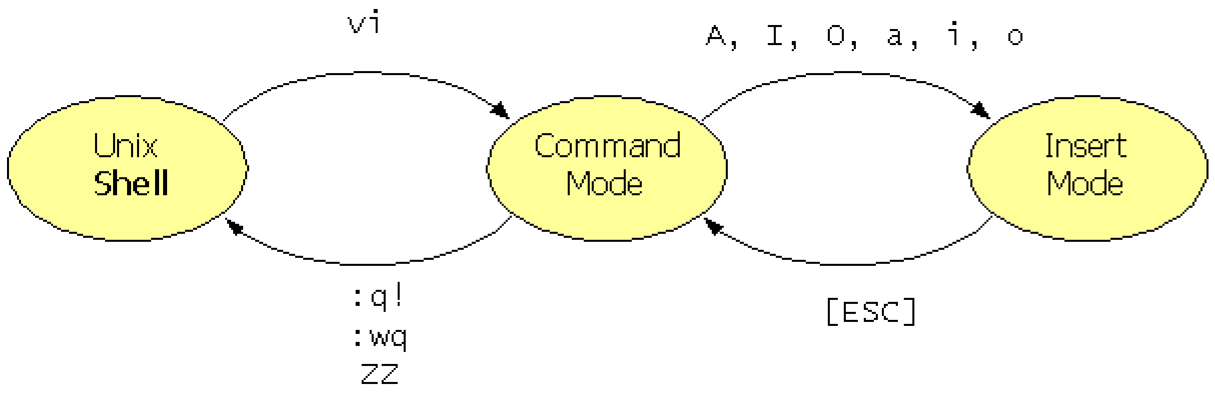
vi Editor:

- ➢ vi is a text editor that is available in Linux and other Unix-like operating systems.
- ➢ vi editor was developed by <u>Bill Joy</u> when he was a student at university of California.
- ➢ It is a <u>command-line text editor</u>, which means that <u>it is used in a terminal window rather than through a graphical user interface</u>.
- ➢ vi is visual editor used to enter and edit text files containing data or documents or programs. It displays the contents of files on the screen and allows the user to add, insert, delete or change parts of the text.
- ➢ <u>**Some of the key features of vi Editor:**</u>
  - <u>Multiple modes:</u> VI has two modes, command mode, and insert mode. In command mode, you can <u>navigate and manipulate text using a variety of commands</u>, and in <u>insert mode, you can type and edit text</u>. **Note:** <u>command mode is default working mode in vi editor</u>.
  - <u>Powerful editing commands</u>: VI provides a variety of powerful editing commands that allow you to quickly and efficiently manipulate text. For example, you can use commands to delete, copy, paste, and move text.
  - <u>Customizable:</u> VI is highly customizable, and you can configure it to suit your preferences and workflow. You can customize key mappings, configure syntax highlighting, and more.
  - <u>Lightweight and fast:</u> VI is a lightweight and fast text editor, which makes it ideal for use on servers and other resource-constrained systems.
  - <u>vi has a steep learning curve:</u> It can take some time to become proficient with it. However, once you have mastered VI, it can be a powerful and efficient tool for editing text in Linux and other Unix-like operating systems.
- ➢ **Syntax to use (open) vi editor:** To work on vi editor we run vi command along with file name as follow**,**

```
vi      filename
```

- ➢ **Working with vi editor:**

Following diagram shows working flow in vi editor.



1. When we run vi command, then it first goes into command mode which is default mode.
2. To switch from command mode to insert mode press any A,a,O,o,I,i key.
3. To exit from insert mode press ESC key. That will again switch back to command mode.
4. To exit from command mode (i.e. exit from vi editor and return back to shell) press :q! or :wq or ZZ

Working modes in vi editor:

- As we know, vi has two modes, viz command mode and insert mode.
- In command mode, we can navigate and manipulate text using a variety of commands and in insert mode, we can type and edit text.
- Let's see these modes in details-

1. Command Mode:

- In this mode all key pressed by the user are interpreted as commands **which are used to navigate or manipulate the text only**. We are **not able to insert any text character in command mode**.
- In command mode there are **reserved (special) keys** when they are pressed will not displayed on the screen but that makes text edition of file.
- In command mode, we can perform following tasks on text such as,
    1. Navigation
    2. Deleting character
    3. Pattern Search
    4. Joining Lines
    5. Copying Lines

Let's see these task in details-

1) Navigation: In vi command mode, to navigate in file following keys are used.

Key	Action
h or Backspace	**Moves cursor left**
l or Space bar	**Moves cursor right.**
k	**Moves cursor up.**
j	**Moves cursor down.**
b	**Moves backward to beginning of word.**
e	**Moves forward to end of word.**
w	**Moves forward to begin of word.**
$	**Moves the cursor to the end of the current line.**
G	**Moves to last line in file. We can also move to specific line in file.E.g. To move 5th line press 5G**
gg	**Moves to first line in file.**

2) **Deleting character:**

- If you want to delete a character in command mode, move the cursor to that character and **press ' x '** the character will disappear and the line will readjust to the change.
- To erase four characters in a row, press 4x and so on…

Eg. **nx Delete n characters**. Here, n is Number of characters to be delete.

3) **String Search (Pattern Search):**

- In command mode, we can also search or locate specific string in file.
- To search any string pattern, search string is given after forward slash '/'
- E.g. To locate the first occurrence of the string "college" simple enter **/college<enter>**.
- To pattern search in file following keys are also used,

Key	Action
n	**Repeat the last search command.**
N	**Repeat the search command in the opposite direction.**
/	**Search forward.**
?	**Search backward.**

4) **Joining Lines:**

- In command mode, we can also join a next line at the end of current line.
- To join next line at the end of current line, ' J ' key is pressed.

5) **Copying Lines:**

- In command mode, we can also copy lines.
- Following keys are used to copy lines-

Key	Action
yy	**Yank or copy the current line (which is to be copied)**
nyy	**Yank or copy 'n' number of lines from current line**
P	**Paste yanked line after or before cursor position**

2) Insert mode (Input Mode):

- Input mode is another mode supported by vi to work on file **basically it used to insert text characters and do other text editing tasks easily**.
- We know that, command is default vi mode but to switch from command mode to insert mode press any A,a,O,o,I,i key.
- Working in an input mode is easy as compared to working in command mode because, this mode permits insertion of new text, editing of existing text or replacement of existing text very easily same as working in notepad of windows.
- **Note:** To exit from input mode we have to press 'ESC' key that will exit insert mode and switch back to command mode.

Exit or Quite from vi editor:

After finishing work in vi editor, we may need to quit Vi editor. We
can save the changes and exit from vi editor by following way-

Key	Action
:w	**Saves file & remains in editing mode.**
:x **OR** **:wq**	**Saves file & quit editing mode.**
:q	**Quit editing mode when no change are made to file.**
:q!	**Quit editing mode but after abandoning changes.**

ZZ	**Save and quit from editing mode.**

vim (vi IMproved) Editor:

- vim is extended and newer version of Vi Editor.
- The original version of vim was developed by Bram Moolenaar, a software engineer from the Netherlands in the early 1990s. The goal behind creation of vim is developing a more powerful and user-friendly text editor.

Features/properties/advantages/characteristics of vim editor:

- Vim (Vi Improved) is a popular command-line open-source text editor which is available on many Unix-like operating systems, including Linux.
- Vim is known for its powerful editing capabilities and its ability to handle large files efficiently.
- One of the key features of Vim is its modal editing interface, which allows users to switch between different modes for different types of editing tasks. For example, in "insert" mode, users can type text directly into the document, while in "command" mode, they can navigate and manipulate the text using variouscommands and shortcuts.
- Vim also supports a wide range of customization options, allowing users to configure the editor to suit their individual needs and preferences. This can include things like custom key mappings, color schemes, and syntax highlighting for different programming languages.
- While Vim has a steep learning curve for new users, it can be a very powerful and efficient tool for those who invest the time to learn it.
- Vim allows screen to be split for editing multiple files.
- Vim can edit files inside compressed archive (.Zip/.tar/.gzip)
- Vim includes support for several popular programming languages (c/c++/Python/Perl)
- Vim includes multilevel undo and redo features
- Vim can also allow to edit files using N/W Protocols such as HTTP, SSH.
- **Syntax to use (open) vim editor:** To work on vim editor we run vim command along with file name as-

```
vim    filename
```

Note: The functionality of vim editor is more advance than vi editor since, vim supports for more features than vi as listed above.

Working modes in vim:

- Basically vim has two "modes": COMMAND mode and INSERT mode.
- In COMMAND mode, you execute commands (like undo, redo, find and replace, quit, etc.) by inputting reserved (special) keys.
- In INSERT mode, you type text same as notepad in windows.
- Also, vim have **VISUAL mode** basically used for text highlighting purpose.
- NOTE: To switch from command mode to VISUAL mode then press v or V or Cntrl+V key from command mode.
- **Working with vim editor:**
  1) When we run vim command, then it first goes into command mode which is default mode in vim also
  2) To switch from command mode to insert mode press any A,a,O,o,I,i key.
  3) To exit from insert mode press ESC key. That will again switch back to command mode.
  4) To exit from command mode (i.e. exit from vim editor and return back to shell) press :q! or :wq or ZZ keys.

Note: The COMMAND mode and INSERT mode of vim is same as vi editor. Hence, we can use same reserved keys of vi in vim also.

Here, we only explain VISUAL mode in details-

VISUAL mode-

- VISUAL mode basically used for text highlighting or text selection purpose.
- If you want to highlight multiple lines or particular words in a line, you can do so in VISUAL mode.
- We enter in visual mode by three different ways explained in below table:

Key	**Action**
v	**enters character-wise visual mode to highlight one character at a time.**
V	**enters line-wise visual mode to highlight entire lines at a time.**
ctrl+v	**enters block-wise visual mode to highlight over rows and columns, etc.**

- Once you enter in VISUAL mode, you can use the keyboard navigation keys (e.g., h,j,k,l or w or b) to highlight a text. Then you can delete (d) a selected text, yank (y) copy selected, or paste (p) copied text.
- To exit from VISUAL mode, use ESC key. It then return back to COMMAND mode.

Shell Programming (Shell Scripting)

- Shell programming, also known as shell scripting, that refers to writing scripts or programs using shell commands in a Unix/Linux shell. We know that, a shell is command-line interpreter that takes user input and executes commands or programs.
- Shell scripts are the files containing a series of shell commands (executable instructions) that can be executed together as a single program. These scripts can be used to automate tasks, perform system maintenance, and create complex programs using basic shell commands and utilities.
- Some of the commonly used shells in Linux include BASH (Bourne-Again SHell), C shell (csh), and Korn shell (ksh). BASH is the default shell in most Linux distributions, and it is widely used for shell programming due to its powerful features.
- To do shell programming, lets discuss basic concepts in shell programming-

Advantages of Shell script or Shell programming:

1) Automation: Shell scripts allow you to automate repetitive tasks, saving you time and effort. You can create a script to perform tasks such as backing up files, updating software, and performing system maintenance.
2) Customization: Shell scripts are highly customizable, and you can modify them to suit your specific needs. You can add or remove commands, change variables, and add conditional statements to create a script that works best for you.
3) Portability: Shell scripts are portable across different Linux systems, so you can use the same script on multiple machines without modification. This can save you time and ensure consistency across your systems.
4) Flexibility: Shell scripts are incredibly flexible, allowing you to **integrate them with other programs** and scripts. You can use them to automate tasks in combination with other programs, or as part of a larger script or application.
5) Script reuse: Once you've created a shell script, you can reuse it multiple times, reducing the time and effort required to perform similar tasks in the future. This makes shell scripts ideal for system administrators and developers who need to perform repetitive tasks regularly.

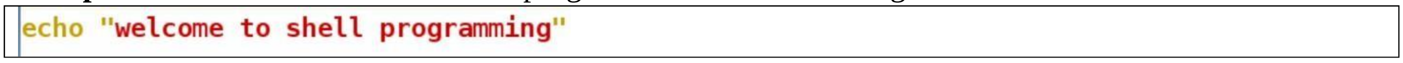
Disadvantages or drawback of Shell script or Shell programming:

1) Steep learning curve: For those new to shell scripting, there may be a steep learning curve to overcome. Shell scripting requires knowledge of command-line syntax and programming concepts, which can be difficult for beginners to grasp.
2) Limited GUI support: Shell scripts are primarily text-based, and while you can use them to automate tasks involving graphical user interfaces (GUIs), they may not be the most efficient or user-friendly solution.
3) Security risks: Shell scripts can pose a security risk if not properly written or executed. A poorly written script can introduce vulnerabilities into your system, or allow unauthorized access to sensitive data.
4) Debugging can be difficult: Debugging shell scripts can be challenging, particularly for complex scripts. Unlike compiled programs, shell scripts are interpreted, which can make it more difficult to pinpoint and fix errors.

5) Not always the most efficient solution: While shell scripts can be an effective way to automate tasks, they may not always be the most efficient solution. Other tools, such as programming languages like Python or Perl, may be better suited to certain tasks.

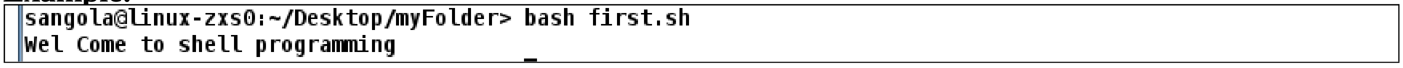
Steps to code & execute shell script (shell program):

1) Create new file using vi command along with **file extension .sh**
2) Code the shell script.
Example- Consider we create new shell program named **first.sh** using vi command as follow-



```
echo "welcome to shell programming"
```

3) Run or execute shell script by using syntax: **bash progName.sh**
Example:



```
sangola@linux-zxs0:~/Desktop/myFolder> bash first.sh
Wel Come to shell programming
```

Shell Variable:

- Shell Variable is an identifier (name given by programmer) which is used to store and manipulate various types of data.
- **Note:** Unlike in some programming languages, **shell variables do not have a memory address** associated with them.
- Syntax to declare and initialize shell variable-
  Syntax: **varibleName=value**
- **Note:** There is no need to specify Data type before variable name.
  Also, Space is NOT allowed either any side of (=) assignment operator during assigning values to variable

Rules To Declare Shell Variable: (Characteristics or Properties of Shell variable)

1) Shell Variable name can contain only letters [a-z,A-Z],Numbers[0-9],Underscore[_].
2) Shell Variable name must be starting either letter or underscore. i.e. shell variable does not start with digit.
3) Space is NOT allowed either any side of (=) assignment operator during assigning values to variable.
4) $(Dollar) sign must be use before variable name to access value of variable.
5) Shell Variable names are case sensitive.
   Ex - num, Num, NUM will be considered as three different variables.

echo Statement:

- echo statement is used to display Message OR Output on terminal screen.
  Syntax: echo "message" OR echo "$variableName"
- **Note:** Give single space after echo.

Example:
1) echo "Welcome to Shell Programming" will display output Welcome to Shell Programming
2) a=95
   echo "$a" will display output 95

read Statement:

- read statement is used to take input values for variables.
  Syntax: read variable1 variable2 variable3

Example:

```
echo  "Enter two value"
read  a    b
echo "FirstVal=$a"
echo "SecondVal=$b"
OUTPUT:
Enter two value
10    20
FirstVal=10
SecondVal=20
```

unset Statement:

- This statement is used to remove the value of variable.
- **Syntax: unset variablename**

Example:

```
x=10
echo "Value of variable before unset= $x"
unset x
echo "Value of variable after unset= $x"
OUTPUT:
Value of variable before unset= 10
Value of variable after unset=
```

Assigning Command O/P to Variable-

- In linux, we can assign output of one command as a value to variable.

Syntax: Variable=$(command [option] [arguments])

Example:

```
x=$(date)
echo  "$x"
OUTPUT:
Mon May 15 10:05:48 IST  2023
```

Commenting Text in LINUX Shell Script:

- As like other programming languages, we can also able to give comment for text present in shell script.
- Basically, comments are used for better readability of source code.
- We can assign comments by two ways. Viz-

1) Single Line Comment:

# symbol is used to give single line comment in shell script that should not be interpreted by shell.

Syntax: # commented Text

2) Multi Line Comment –

- We can give multi line comment using **:** ' to Start commenting and ' to Stop commenting.
- **Example:**

```
: ' First line
  Second line
  Third line '
```

Note: In multi-line comment there is space between start comment as **: <space> '**

expr command:

- The expr command in linux is used to evaluate a given expression and displays its corresponding output.
- Basically expr command is used to perform:

1) Basic operations like addition, subtraction, multiplication, division, and modulus on integers.
2) Evaluating string operations like substring, length of strings etc.

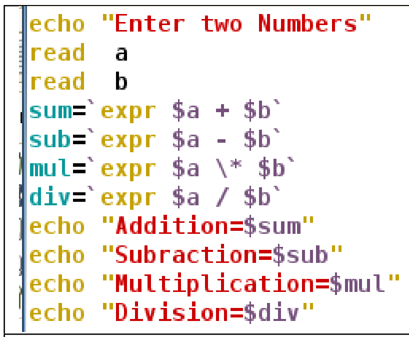
Syntax: `expr<space>$var1<space>Operator<space>$var2`

In above syntax,

- var1 and var2 are the variables.
- Operator is any arithmetic operator
- Entire expression must enclosed within backtick (`)
- Compulsory mention space at <space>

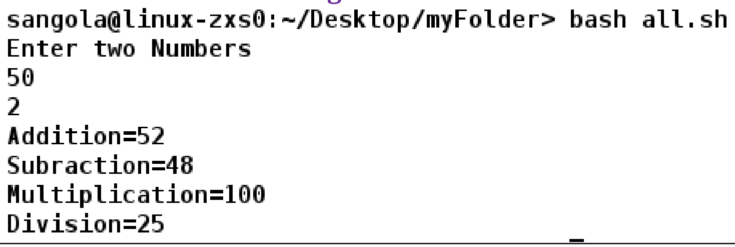
1) Performing Basic Arithmetic Operations-

We create **all.sh** file as follow- (Using **vi all.sh** command)



```
echo "Enter two Numbers"
read  a
read  b
sum=`expr $a + $b`
sub=`expr $a - $b`
mul=`expr $a \* $b`
div=`expr $a / $b`
echo "Addition=$sum"
echo "Subraction=$sub"
echo "Multiplication=$mul"
echo "Division=$div"
```

We execute all.sh file and got OUTPUT:



```
sangola@linux-zxs0:~/Desktop/myFolder> bash all.sh
Enter two Numbers
50
2
Addition=52
Subraction=48
Multiplication=100
Division=25
```

Note:

- In multiplication if we use only * operator then it will treat as one of metacharacter (special character having special meaning in shell like < , > , `, |,$ etc) in shell and we will not get multiplication result. Since, to perform multiplication, it is escaped by \ so that the shell does NOT interprets it as metacharacter.
- A space on either side of operator is essential in expr command.
- expr can handle only integers so division result returns only integral value, to obtain float result use **bc command with pipe operator**. Ex- **echo $a / $b | bc -l**
- expr is an external program used by Bourne shell. It uses expr external program with the help of backtick. The backtick(`) is actually called <u>command substitution</u>.

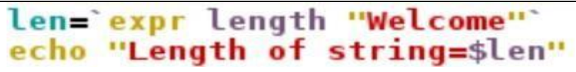
2) Evaluating string operations:

2.1) Finding Length of string:

Using expr command we can also find length of string (number of characters in a string) as follow.

<u>Syntax</u>- **`expr length $stringvariable`**

Example:



```
len=`expr length "Welcome"`
echo "Length of string=$len"
```

OUTPUT:

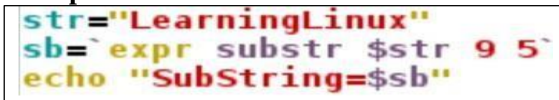
Length of string=7

2.2) Extract Sub-string from main string:

Using expr command we can extract substring from main string.

<u>Syntax-</u> **`expr substr $stringvariable StartIndexvalue LengthValue`**

Example:



```
str="LearningLinux"
sb=`expr substr $str 9 5`
echo "SubString=$sb"
```

OUTPUT:

SubString=Linux

Note: Do not assign multi-word string to string variable.

test Command:

➢ The test command in LINUX used to-
- Check a given condition (comparing two values)
- Comparing strings.
- Performing logical operations
- Performing file test operations.

Syntax: test $va1 operator $var2

➢ **Note: (it is contradict with other programming language)**
- test command returns 1 if condition is FALSE
- test command returns 0 if condition becomes TRUE

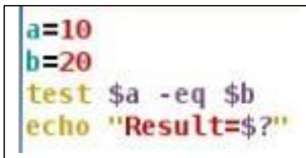
➢ Linux operating system also have several categories of operators which are discussed below.

1) Comparison Operators:

Comparison operators are used to compare values. The list of comparison operators are shown in following table-

Comparison Operators		
-eq	**INTEGER1 is equal to INTEGER2**	**INTEGER1 -eq INTEGER2**
-ge	**INTEGER1 is greater than or equal to INTEGER2**	**INTEGER1 -ge INTEGER2**
-gt	**INTEGER1 is greater than INTEGER2**	**INTEGER1 -gt INTEGER2**
-le	**INTEGER1 is less than or equal to INTEGER2**	**INTEGER1 -le INTEGER2**
-lt	**INTEGER1 is less than INTEGER2**	**INTEGER1 -lt INTEGER2**
-ne	**INTEGER1 is not equal to INTEGER2**	**INTEGER1 -ne INTEGER2**

Examples:



```
a=10
b=20
test $a -eq $b
echo "Result=$?"
```

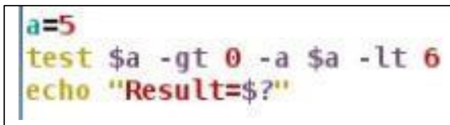
OUTPUT:
Result=1
Note: Since, condition is False hence it return 1
? holds result is of previous test command

2) Logical Operators:

Comparison operators are used to compare values. The list of comparison operators are shown in following table-

Operator	**Description**
!	This is logical negation. This inverts a true condition into false and vice versa.
-o	This is logical **OR**. If one of the operands is true, then the condition becomes true.
-a	This is logical **AND**. If both the operands are true, then the condition becomes true otherwise false.

Example:



```
a=5
test $a -gt 0 -a $a -lt 6
echo "Result=$?"
```

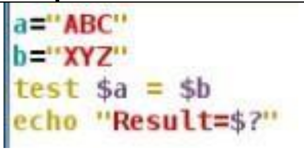
OUTPUT:
Result=0
Note: Both conditions are true hence it returns 0

3) String Comparison Operators:

- Following table shows some operators which are used to compare strings also,

Operator	Description	Example
Consider we have two string variables- a="ABC" and b="XYZ"		
=	Checks if the value of two operands is equal or not; if yes, then the condition becomes true.	[$a = $b] is not true.
!=	Checks if the value of two operands is equal or not; if values are not equal then the condition becomes true.	[$a != $b] is true.
-z	Checks if the given string operand size is zero; if it is zero length, then it returns true.	[-z $a] is not true.

Example:



```
a="ABC"
b="XYZ"
test $a = $b
echo "Result=$?"
```

OUTPUT:

Result=0

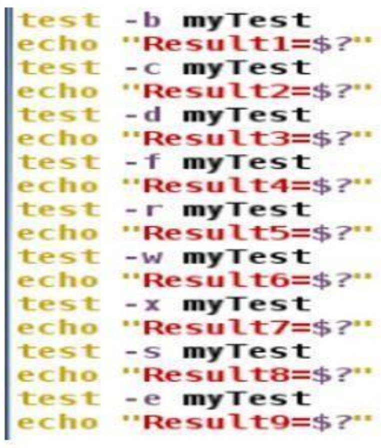
4) File Test Operators:

- We have a few operators that can be used to test various properties associated with a LINUX file.
- Assume we have file variable **file** which holds an existing file named "**myTest**" the size of file is 100 bytes and has read, write and execute permissions are ON. Then we discuss file test operators as follow-

Operator	Description	Example
-b	Checks if file is a block special file; if yes, then the condition becomes true.	[-b $file] is false.
-c	Checks if file is a character special file; if yes, then the condition becomes true.	[-c $file] is false.
-d	Checks if file is a directory; if yes, then the condition becomes true.	[-d $file] is false.
-f	Checks if file is an ordinary/regular file as opposed to a directory or special file; if yes, then the condition becomes true.	[-f $file] is true.
-r	Checks if file is readable; if yes, then the condition becomes true.	[-r $file] is true.
-w	Checks if file is writable; if yes, then the condition becomes true.	[-w $file] is true.
-x	Checks if file is executable; if yes, then the condition becomes true.	[-x $file] is true.
-s	Checks if file has size greater than 0; if yes, then condition becomes true.	[-s $file] is true.
-e	Checks if file exists; is true even if file is a directory but exists.	[-e $file] is true.

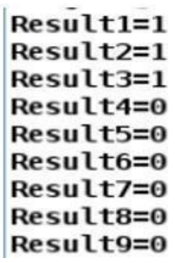
Note: Block special files are provided for logical volumes and disk devices on the operating system and are solely for system use in managing file systems, paging devices, and logical volumes. These files should not be used for other purposes.

Example: We create fileTest.sh file as-



```
test -b myTest
echo "Result1=$?"
test -c myTest
echo "Result2=$?"
test -d myTest
echo "Result3=$?"
test -f myTest
echo "Result4=$?"
test -r myTest
echo "Result5=$?"
test -w myTest
echo "Result6=$?"
test -x myTest
echo "Result7=$?"
test -s myTest
echo "Result8=$?"
test -e myTest
echo "Result9=$?"
```

OUTPUT:



```
Result1=1
Result2=1
Result3=1
Result4=0
Result5=0
Result6=0
Result7=0
Result8=0
Result9=0
```

Conditional Statements:

- Linux shell script also have conditional statements to test different conditions which are explained as follow.
- There are total 5 conditional statements which can be used in linux bash programming
  1) if statement
  2) if-else statement
  3) if..elif..else..fi statement (Else If ladder)
  4) if..then..else..if..then..fi..fi..(Nested if)
  5) switch statement

1) if statement:

- It is conditional statement or decision making statement.
- The block of if executes if and only if specified condition is true. If condition is false then block of if not executes.

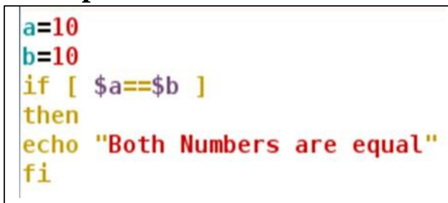
Syntax:

```
        if [<space> condition <space>]
        then
                Statements1
                Statements2
                 ……
                Statement N
        fi
```

In above syntax,

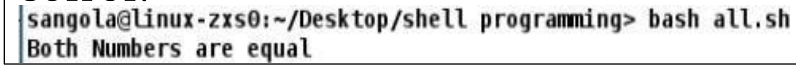
- **<space> must be mentioned within square bracket.**
- **then must be mention to next line of if**
- **fi is used for end of if**

Example: consider we create a shell program named **all.sh** with vi command as follow-



```
a=10
b=10
if [ $a==$b ]
then
echo "Both Numbers are equal"
fi
```

OUTPUT:



```
sangola@linux-zxs0:~/Desktop/shell programming> bash all.sh
Both Numbers are equal
```

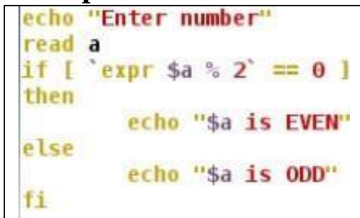
2) if-else statement:

- if-else is also conditional statement or decision making statement.
- The block of if executes if and only if specified condition is true. If condition is false then block of else is executed.

Syntax:

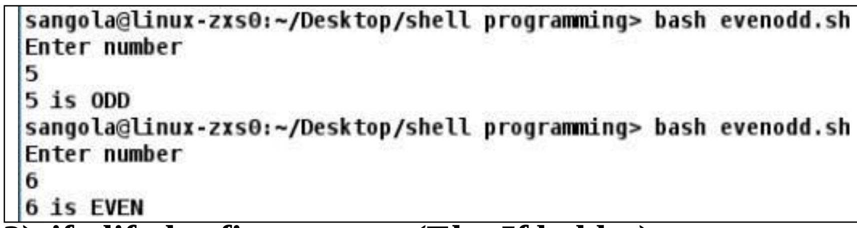
```
        if  [<space> condition <space>]

        then
                If block Statements
        else
                Else block Statements
        fi
```

Example: We create a shell program named **evenodd.sh** with vi command as follow-



```
echo "Enter number"
read a
if [ `expr $a % 2` == 0 ]
then
        echo "$a is EVEN"
else
        echo "$a is ODD"
fi
```

OUTPUT:



```
sangola@linux-zxs0:~/Desktop/shell programming> bash evenodd.sh
Enter number
5
5 is ODD
sangola@linux-zxs0:~/Desktop/shell programming> bash evenodd.sh
Enter number
6
6 is EVEN
```

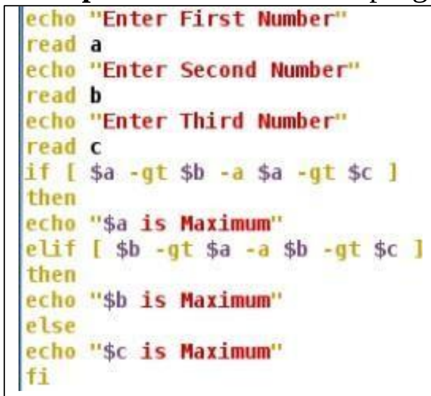
3) if-elif-else-fi statement (Else If ladder):

- This is another way of if-else statement where we are able to check series of conditions one after another.
- The functionality of else-if ladder is given in following syntax.

Syntax:

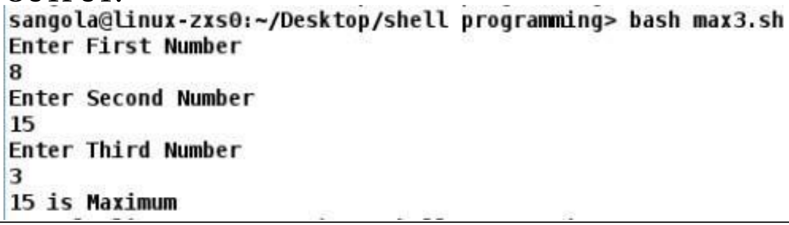
```
if  [<space> condition1 <space>]

then
        Statements block 1
elif [<space> condition2 <space>]
then
        Statements block 2
elif [<space> condition3 <space>]
then
        Statements block 3
else

        Else block Statements
fi
```

Example: We create a shell program named **max3.sh** with vi command as follow-



```
echo "Enter First Number"
read a
echo "Enter Second Number"
read b
echo "Enter Third Number"
read c
if [ $a -gt $b -a $a -gt $c ]
then
echo "$a is Maximum"
elif [ $b -gt $a -a $b -gt $c ]
then
echo "$b is Maximum"
else
echo "$c is Maximum"
fi
```

OUTPUT:



```
sangola@linux-zxs0:~/Desktop/shell programming> bash max3.sh
Enter First Number
8
Enter Second Number
15
Enter Third Number
3
15 is Maximum
```

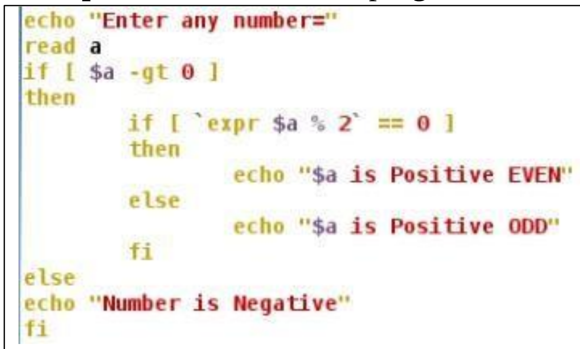
4) if-then-else-if-then-fi-fi :(Nested if)

- In nested form of if statement, we can define another block of if and else statement inside of existing if or else statement block.
- The functionality of nested if statement is given in following syntax.

Syntax:

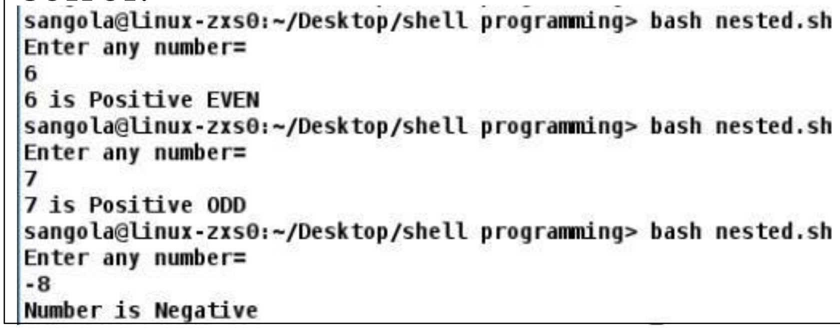
```
if  [<space> condition1 <space>]

then
        if [<space> condition2 <space>]

        then

                Statements block 1
        else
                Statements block 2
        fi
elif [<space> condition2 <space>]
then
        if [<space> condition3 <space>]

        then

                Statements block 3
        else
                Statements block 4
        fi
else

        Else block Statements
fi
```

Example: We create a shell program named **nested.sh** with vi command as follow-



```
echo "Enter any number="
read a
if [ $a -gt 0 ]
then
        if [ `expr $a % 2` == 0 ]
        then
                echo "$a is Positive EVEN"
        else
                echo "$a is Positive ODD"
        fi
else
echo "Number is Negative"
fi
```

OUTPUT:



```
sangola@linux-zxs0:~/Desktop/shell programming> bash nested.sh
Enter any number=
6
6 is Positive EVEN
sangola@linux-zxs0:~/Desktop/shell programming> bash nested.sh
Enter any number=
7
7 is Positive ODD
sangola@linux-zxs0:~/Desktop/shell programming> bash nested.sh
Enter any number=
-8
Number is Negative
```

5) case..esac (switch case) statement:

- ➢ The control statement that allows us to make a decision from number of choices is called "switch statement" or "switch-case-default" statement.
- ➢ The switch statement is alternative for else-if ladder where we check multiple conditions.
- ➢ The functionality of case esac statement is given in following syntax.

Syntax:

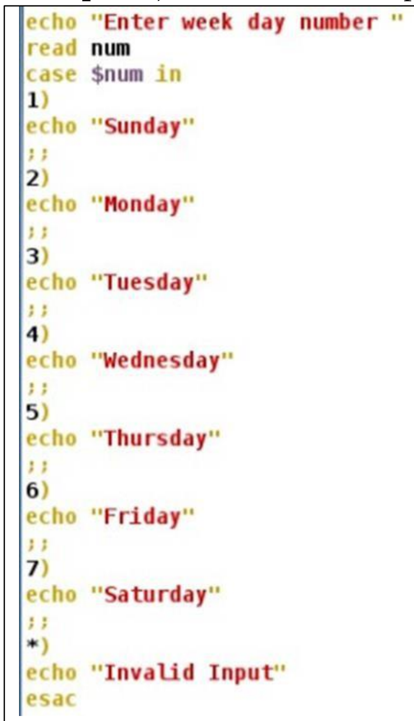
```
        case Expression in
          pattern1)
            Statement(s) to be executed if pattern1 matches with Expression
            ;;
          pattern2)
            Statement(s) to be executed if pattern2 matches with Expression
            ;;
          pattern3)
            Statement(s) to be executed if pattern3 matches with Expression
            ;;
```

```
    *)
      Default condition to be executed if no matches with Expression
      ;;
  esac
```

In above syntax:

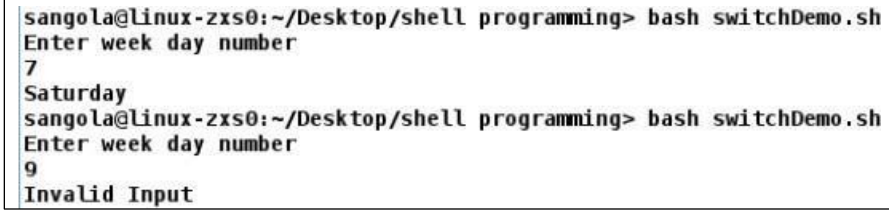
- **Expression** is compared against every pattern until a match is found. The statement(s) following the matching pattern executes.
- When statement(s) part executes, the command ;; indicates that the program flow should jump to the end of the entire case statement. This is similar to break in the C programming language
- If we forgot to mention ;; then control jump to next pattern block.
- **case, in** and **esac** are reserve word.

Example: 1) We create a shell program named **switchDemo.sh** with vi command as follow-



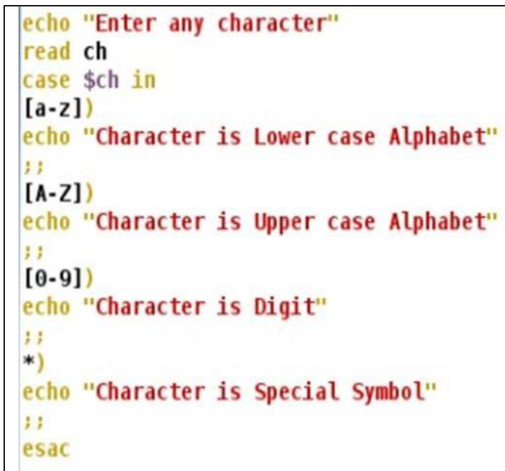
```
echo "Enter week day number "
read num
case $num in
1)
echo "Sunday"
;;
2)
echo "Monday"
;;
3)
echo "Tuesday"
;;
4)
echo "Wednesday"
;;
5)
echo "Thursday"
;;
6)
echo "Friday"
;;
7)
echo "Saturday"
;;
*)
echo "Invalid Input"
esac
```

OUTPUT:



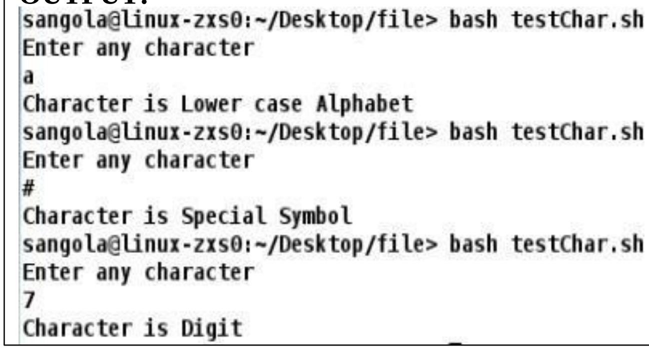
```
sangola@linux-zxs0:~/Desktop/shell programming> bash switchDemo.sh
Enter week day number
7
Saturday
sangola@linux-zxs0:~/Desktop/shell programming> bash switchDemo.sh
Enter week day number
9
Invalid Input
```

2) We create a shell program named **testChar.sh** with vi command as follow-



```
echo "Enter any character"
read ch
case $ch in
[a-z])
echo "Character is Lower case Alphabet"
;;
[A-Z])
echo "Character is Upper case Alphabet"
;;
[0-9])
echo "Character is Digit"
;;
*)
echo "Character is Special Symbol"
;;
esac
```

OUTPUT:



```
sangola@linux-zxs0:~/Desktop/file> bash testChar.sh
Enter any character
a
Character is Lower case Alphabet
sangola@linux-zxs0:~/Desktop/file> bash testChar.sh
Enter any character
#
Character is Special Symbol
sangola@linux-zxs0:~/Desktop/file> bash testChar.sh
Enter any character
7
Character is Digit
```

Looping Statements:

Introduction:

- We know that to develop any program there are three kinds of logics were used viz (namely) Sequence logic, Selection logic or Iteration logic.
- In Iteration logic (Looping) is process in which block of statement is repeatedly executed till the condition is true and exit from the loop when condition becomes false.

Every loop having two segments (parts) viz-

1) Body of loop 2) Control statement (Condition)

- The control statement tests the condition and executes body of loop repeatedly till the condition is true. If condition becomes false then loop terminates.
- By Using loops, we can execute a set of commands repetitively until condition becomes false.
- Linux shell have following looping statements-

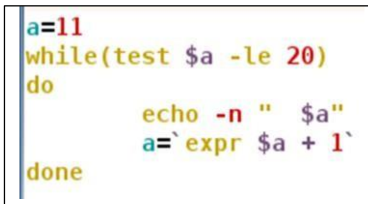
1) while statement:

- ➢ The while loop is entry-controlled loop i.e. counter condition is checked first, if it is True then body of loop is executed repeatedly till condition is True. If condition becomes False then control exit from the body of loop.
- ➢ while loop is simplest loop in all looping statements in linux shell. The general form of while loop is as fallow:

Syntax:

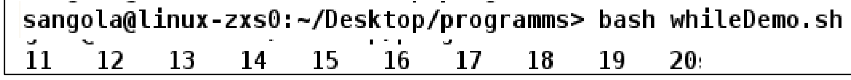
```
while(Condition) do
        Instruction1 Instruction2
         .
         .
        InstructionN
done
```

Example: We create a shell program named **whileDemo.sh** that prints series 11 to 20 using while loop.



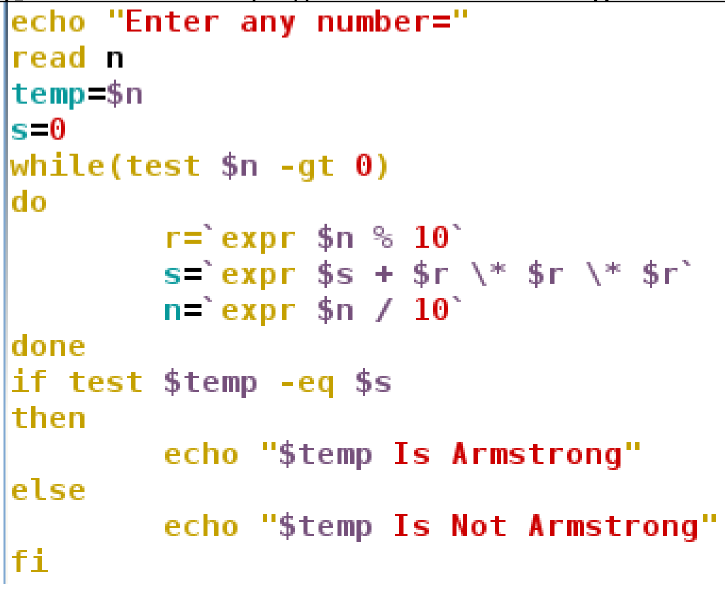
```
a=11
while(test $a -le 20)
do
	echo -n "  $a"
	a=`expr $a + 1`
done
```

OUTPUT:



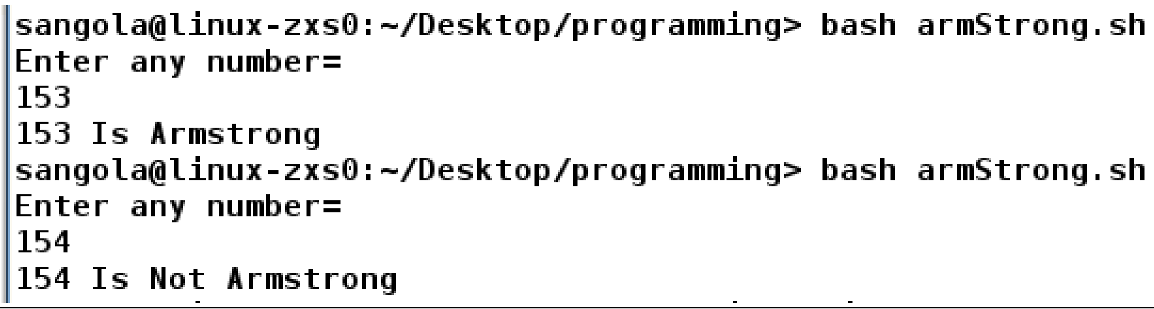
```
sangola@linux-zxs0:~/Desktop/programms> bash whileDemo.sh
 11   12   13   14   15   16   17   18   19   20
```

2) We create a shell program named **armStrong.sh** that check entered number is Armstrong or not.



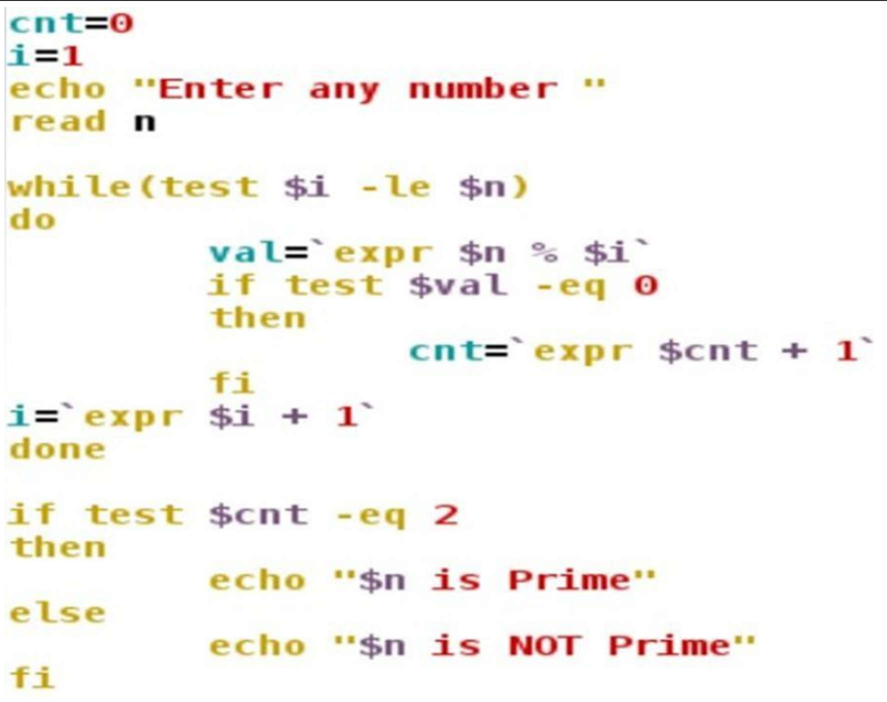
```
echo "Enter any number="
read n
temp=$n
s=0
while(test $n -gt 0)
do
	r=`expr $n % 10`
	s=`expr $s + $r \* $r \* $r`
	n=`expr $n / 10`
done
if test $temp -eq $s
then
	echo "$temp Is Armstrong"
else
	echo "$temp Is Not Armstrong"
fi
```

OUTPUT:



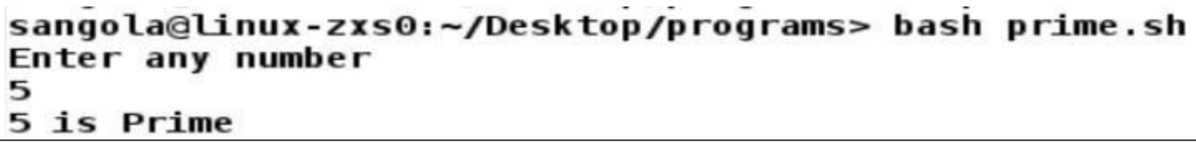
```
sangola@linux-zxs0:~/Desktop/programming> bash armStrong.sh
Enter any number=
153
153 Is Armstrong
sangola@linux-zxs0:~/Desktop/programming> bash armStrong.sh
Enter any number=
154
154 Is Not Armstrong
```

3) Following **prime.sh** program checks whether number is Prime or not.



```
cnt=0
i=1
echo "Enter any number "
read n

while(test $i -le $n)
do
        val=`expr $n % $i`
        if test $val -eq 0
        then
                cnt=`expr $cnt + 1`
        fi
i=`expr $i + 1`
done

if test $cnt -eq 2
then
        echo "$n is Prime"
else
        echo "$n is NOT Prime"
fi
```

OutPut:



```
sangola@linux-zxs0:~/Desktop/programs> bash prime.sh
Enter any number
5
5 is Prime
```

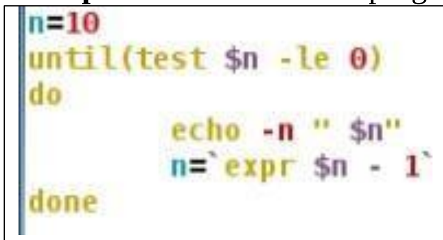
2) until loop:

- The until loop statement is opposite to while loop statement because, until loop body is executed repetitively as long as condition remains FALSE.

Syntax:

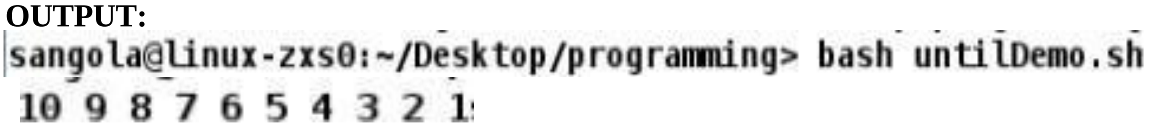
```
        until (condition)
        do
                Instruction1
                Instruction2
                .
                .
                InstructionN
        done
```

Example: We create a shell program named **untilDemo.sh** that prints series 10 to 1 using until loop.



```
n=10
until(test $n -le 0)
do
        echo -n " $n"
        n=`expr $n - 1`
done
```

OUTPUT:



```
sangola@linux-zxs0:~/Desktop/programming> bash untilDemo.sh
 10 9 8 7 6 5 4 3 2 1
```

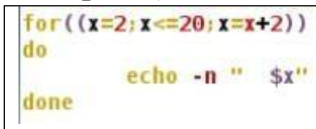
3) for loop:

- The for loop is also <u>entry controlled</u> loop i.e. counter condition is checked first, if it is True then body of loop is executed repeatedly till condition is True. If condition becomes False then control exit from the body of loop.
- In linux, we can use for loop similar to 'C' language having following syntax

Syntax:

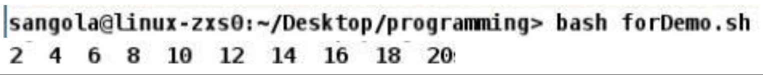
```
for (( initialization of counter ; test counter condition ; increment or decrement counter ))
do
        Body of loop ;
done
```

Example: 1) We create a shell program named **forDemo.sh** that prints series like 2 4 6 8 10 . . . 20



```
for((x=2;x<=20;x=x+2))
do
        echo -n "  $x"
done
```

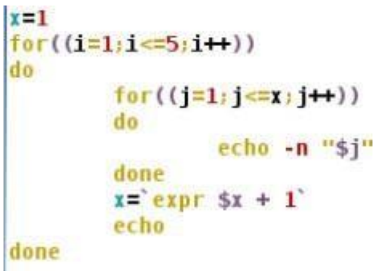
OUTPUT:



```
sangola@linux-zxs0:~/Desktop/programming> bash forDemo.sh
 2  4  6  8  10  12  14  16  18  20
```

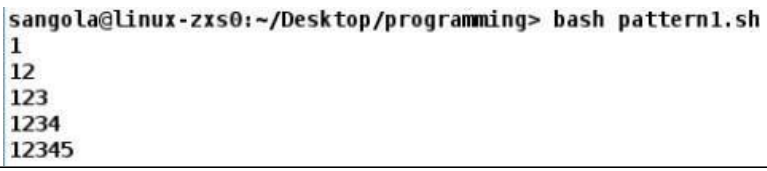
2) Following shell program named **pattern1.sh** print the pattern-

To print pattern we use **nesting of for loop** as follow-



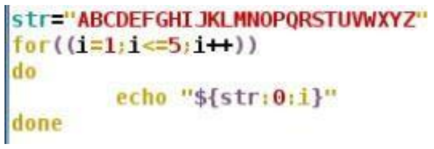
```
x=1
for((i=1;i<=5;i++))
do
        for((j=1;j<=x;j++))
        do
                echo -n "$j"
        done
        x=`expr $x + 1`
        echo
done
```

Output:



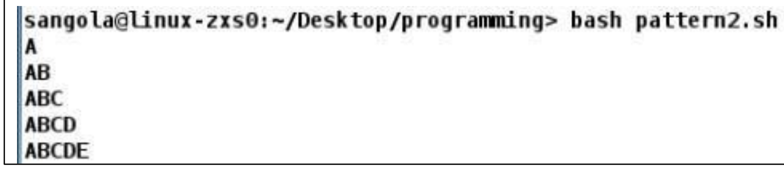
```
sangola@linux-zxs0:~/Desktop/programming> bash pattern1.sh
1
12
123
1234
12345
```

3) Following shell program named **pattern2.sh** print the pattern-



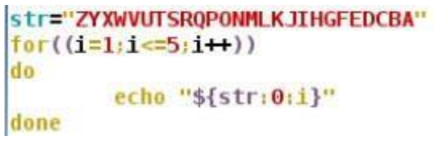
```
str="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
for((i=1;i<=5;i++))
do
        echo "${str:0:i}"
done
```

OutPut:



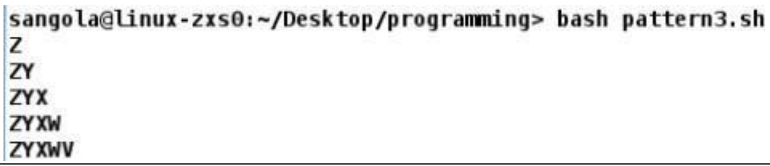
```
sangola@linux-zxs0:~/Desktop/programming> bash pattern2.sh
A
AB
ABC
ABCD
ABCDE
```

4) Following shell program named **pattern3.sh** print the pattern-



```
str="ZYXWVUTSRQPONMLKJIHGFEDCBA"
for((i=1;i<=5;i++))
do
        echo "${str:0:i}"
done
```

Output:



```
sangola@linux-zxs0:~/Desktop/programming> bash pattern3.sh
Z
ZY
ZYX
ZYXW
ZYXWV
```

4) for in loop: (for each loop)

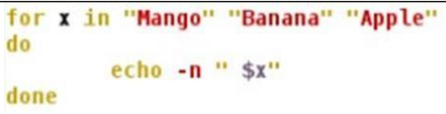
- ➢ The for in loop operate on lists of items.
- ➢ It repeats a set of commands for every item in a list.

Syntax:

```
for var in word1 word2...wordN
do
        Statement to be executed
done
```

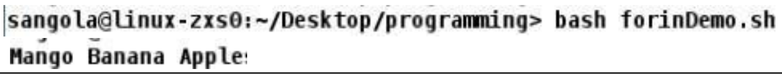
In above syntax-
var is the name of a variable and word1 to wordN are sequences of characters separated by spaces (words).
Each time the for loop executes, the value of the variable var is set to the next word in the list of words, word1 to wordN.

Example- We create a shell program named **forinDemo.sh** that prints words-



```
for x in "Mango" "Banana" "Apple"
do
        echo -n " $x"
done
```

OUTPUT:



```
sangola@linux-zxs0:~/Desktop/programming> bash forinDemo.sh
 Mango Banana Apple
```

Input/Output Redirection:

- ➢ Input/output redirection in Linux refers to the process of changing the source or destination of input or output streams from the default options.
- ➢ It allows you to control where the input comes from and where the output goes when executing commands in the terminal.
- ➢ In linux shell, by default input comes from keyboard and output (result) goes towards monitor.
- ➢ In Linux, the following symbols are commonly used for input/output redirection:

1) ">" (greater-than symbol): This symbol redirects the standard output of a command to a file, overwriting the file if it already exists.

Syntax: **command > file**

Example:

Rps@linux~/Desktop/Myfold> date > myfile

- The above command **writes system current date** to **myfile** file.

2) ">>" (double greater-than symbol): This symbol redirects the standard output of a command to a file, but appends the output to the end of the file if it already exists.

Syntax: **command >> file**

Example:

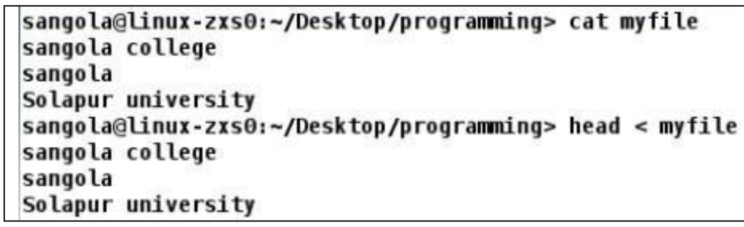
Rps@linux~/Desktop/Myfold> cat >> info

- The above command appends extra data to existing **info** file.

3) "<" (less-than symbol): Using < symbol, the command reads its input from the specified file.

Syntax: **command < file**

Example: Consider, we have **myfile** as follow-



```
sangola@linux-zxs0:~/Desktop/programming> cat myfile
sangola college
sangola
Solapur university
sangola@linux-zxs0:~/Desktop/programming> head < myfile
sangola college
sangola
Solapur university
```

4) "2>" (greater-than symbol followed by the number 2): This symbol redirects the standard error (stderr) output of a command to a file. It is often used to separate error messages from regular output. Also, it is helpful to maintain error log file while installing a software.

Syntax: command 2> error.txt

Example:

```
Rps@linux~/Desktop/Myfold> datedd 2> errorFile
```

The above, command would write error message **'datedd: command not found'** to **errorFile.**

5) "|" (pipe symbol): This symbol redirects the output of one command to another command as input. It allows you to chain multiple commands together.

Syntax: command1 | command2

Example: consider we have file **myData** file as-

```
Rps@linux~/Desktop/MyFold> cat   myData
linux is open source operating system.
Rps@linux~/Desktop/MyFold> cat   myData | tr "a-z"   "A-Z"
LINUX IS OPEN SOURCE OPERATING SYSTEM.
```

Piping in Linux-

- Piping in Linux refers to the process of connecting the output of one command to the input of another command using the pipe symbol (|).
- It allows you to chain multiple commands together, where the output of the preceding command serves as the input for the next command.
- When you use the pipe symbol (|) to create a pipeline, the output produced by the first command is passed directly as input to the second command, and so on if there are more commands in the pipeline. This enables you to perform complex operations by combining the functionality of multiple commands.
- **Syntax: command1 | command2**

In above syntax, the output of command1 is piped to command2. The output produced by command1 becomes the input for command2

Example: consider we have file **myData** file as-

```
Rps@linux~/Desktop/MyFold> cat   myData
linux is open source operating system.
Rps@linux~/Desktop/MyFold> cat   myData | tr "a-z"   "A-Z"
LINUX IS OPEN SOURCE OPERATING SYSTEM.
```

We can also, chain multiple commands together as follow-

Syntax: command1 | command2 | command3 | command4

In above syntax, the output of command1 is piped to command2, the output of command2 is piped to command3, and so on.

Example-

```
Rps@linux~/Desktop/MyFold> ls | find *.sh | wc -l
15
```

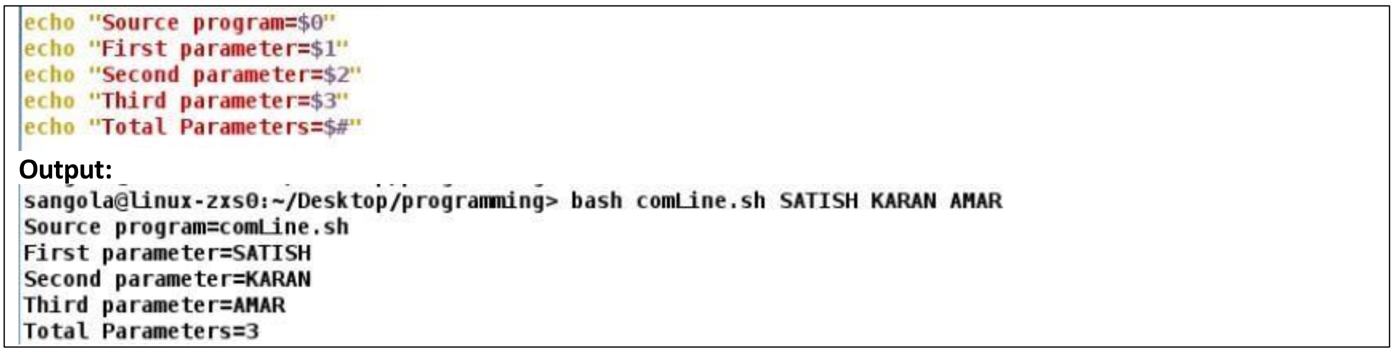
In above example, the **output of ls command (it list the files and directories) is piped to find command (it finds all .sh files), the output of find command is piped to wc command (it counts all .sh files)**.

Command Line Arguments for shell program-

- Like other programming languages, we can also pass arguments for shell program referred as "command line arguments"
- The first passed argument is stored at index 0 and we must have to pass source shell program name.

- ➢ The second passed argument is stored at index 1.
- ➢ The third passed argument is stored at index 2.
- ➢ The fourth passed argument is stored at index 3. And so on..
- ➢ We can access value of passed argument with $ symbol along with index value. Such as $0 will access source shell program name, $1 will access value of second pass argument & so on..
- ➢ $# will access total number of passed arguments.
- ➢ Syntax to pass argument to shell program:
    - **bash sourcePrg.sh arg1 arg2 arg3**
- ➢ Following program demonstrate command line arguments for shell program-

Example 1) Following shell program named **comLine.sh** shows command line arg. concept as follow-

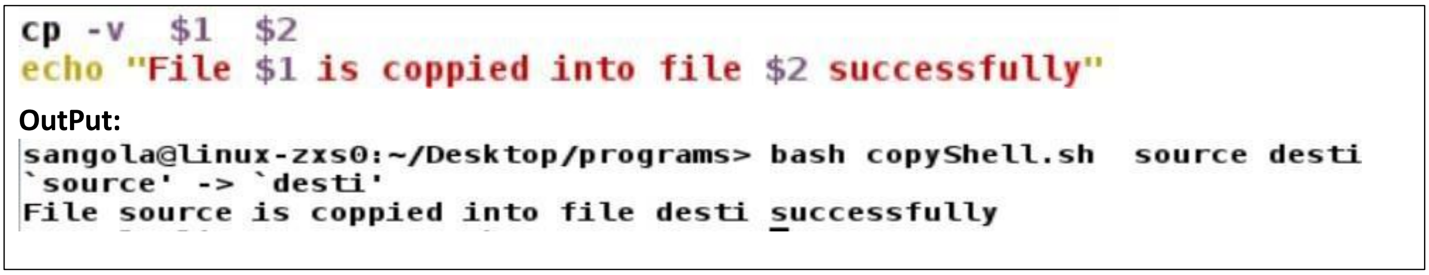


```
echo "Source program=$0"
echo "First parameter=$1"
echo "Second parameter=$2"
echo "Third parameter=$3"
echo "Total Parameters=$#"
```

Output:

```
sangola@linux-zxs0:~/Desktop/programming> bash comLine.sh SATISH KARAN AMAR
Source program=comLine.sh
First parameter=SATISH
Second parameter=KARAN
Third parameter=AMAR
Total Parameters=3
```

Example 2) Shell program named **copyShell.sh** that copy the content of file into another using command line argument concept.



```
cp -v  $1  $2
echo "File $1 is coppied into file $2 successfully"
```

OutPut:

```
sangola@linux-zxs0:~/Desktop/programs> bash copyShell.sh  source desti
`source' -> `desti'
File source is coppied into file desti successfully
```

Metacharacters:

- ➢ In Linux, metacharacters are <u>special characters having special meanings or usage when used in command-line interfaces or shell scripts</u>.
- ➢ These characters are <u>used to modify or expand the behavior of commands or to create complex patterns for searching and manipulating files</u>.
- ➢ Following table shows some metacharacters with meaning and example-

Metacharacters	Usage And examples
< Redirection	**Get input for the command. <u>Example:</u>** **sort < one.txt** **Print out the contents of one.txt With the lines sorted.**
> Redirection	**Send the output of the command into the file. If the file does not exist, create it. If it does exist, overwrite it.** **<u>Example:</u>** **cal > one.txt** **puts a current calendar into one.txt**

>> Redirection	Send the output of the command to the end of the file. If the file does not exist, it will be created. If it does exist, it will be appended. <u>Example:</u> date >> one.txt This adds the current date and time to the end of one.txt
| Pipe	This sends the output of the one command to the input of another command. <u>Example:</u> cat one.txt | grep linux print the lines in one.txt that contain the string "linux"
? Matching	Match any single character. <u>Example:</u> ls e?.txt list all files that start with e, end with .txt and have any 1 character in between (like e2.txt)
* Matching	Matches any number of any characters. <u>Example:</u> ls e*.txt This will list all files that start with e, end with .txt and have any characters in between.(like e12xyz.txt)
[ ] Matching	One of the characters within the square brackets must be matched. <u>Example:</u> ls -l e[abc].txt This will give a long listing of ea.txt, eb.txt and/or ec.txt
$	Denotes that a string is a variable name - Not used when assigning a variable. <u>Example:</u> my_variable="this string" # Assigning the variable echo $my_variable Referring to the contents of the variable
\ Escape	Ignore the shell's special meaning for the character after this symbol, Treat it as though it is just an ordinary character. <u>Example:</u> echo "I have \$300.00" Print the $ instead of using it to look up a variable name. Also used to put commands on multiple lines. <u>Example:</u> cat filename.txt \ # Don't run yet... date \ # Still don't do anything... cal # No backslash on the last one, so run all the commands
( ) Grouping commands	If you want to run two commands and send their output to the same place, put them in a group together. <u>Example:</u> (cal; date) > filename.txt Put a calendar and the date in filename.txt
" Double Quoting	Used to group strings that contain spaces and other special characters. <u>Example:</u> my_variable="test" Assign a string to the variable
	Used to prevent the shell from interpreting special characters within the quoted string. <u>Example:</u>

' Single Quoting	my_variable='backslash: \' Assign the string to the variable
` Back tick	Used within a quoted string to force the shell to interpret and run the command between the backticks. Example: my_variable="date is: `date`" Store the string AND the output of the date command
&& double ampersand	Run the command to the right of the double -ampersand ONLY IF the command on the left succeeded in running. Example: mkdir SYECS && echo "Made the directory" Print a message on success of the mkdir command
|| Double pipe	Run the command on the right of the double pipe ONLY IF the command on the left failed. Example: mkdir SYECS || echo "mkdir failed!" Print a message on failure of the mkdir command
;	Allows you to list multiple commands on a single line, separated by this character. Example: date;cal;who Print the date, calendar and currently logged in users
= Assignment	Set the variable named on the left to the value presented on the right. Example: my_variable="Hello World!" no space between the variable name and the string.