SCRIPTING LANGUAGE

-N HEMA NAGA VAISHNAVI

Introduction

- **Introduction to computer systems**:

A computer is an electronic device that can be programmed to accept data (input), process it and generate result (output). A computer along with additional hardware and software together is called a computer system. A computer system primarily comprises a central processing unit (CPU), memory, input/output devices and storage devices. All these components function together as a single unit to deliver the desired output. A computer system comes in various forms and sizes. It can vary from a high-end server to personal desktop, laptop, tablet computer, or a smartphone.

- **Introduction to OS**

All the computer systems have OS installed, it is a program that acts as an Interface between the system hardware and the user making the tasks easier. It is important software which runs on a computer and controls the set of instructions and wisely utilizes each part of the Computer.

- **Types of OS**
  - **Windows:** It is also used in many offices because it gives you access to productivity tools such as calendars, word processors, and spreadsheets. It is a graphic user interface.
  - **Linux:** Server OS for web servers, database servers, file servers, email servers and any other type of shared server. Designed to support high-volume and multithreading applications, Linux is well-suited for all types of server applications. Desktop OS for personal productivity computing. It is command line interface.
  - **Mac:** It works hand in hand with iCloud to keep photos, documents and other stuff up to date on all your devices. It makes your Mac work like magic with your iPhone and other Apple devices.

- **Difference between Windows and Linux OS**

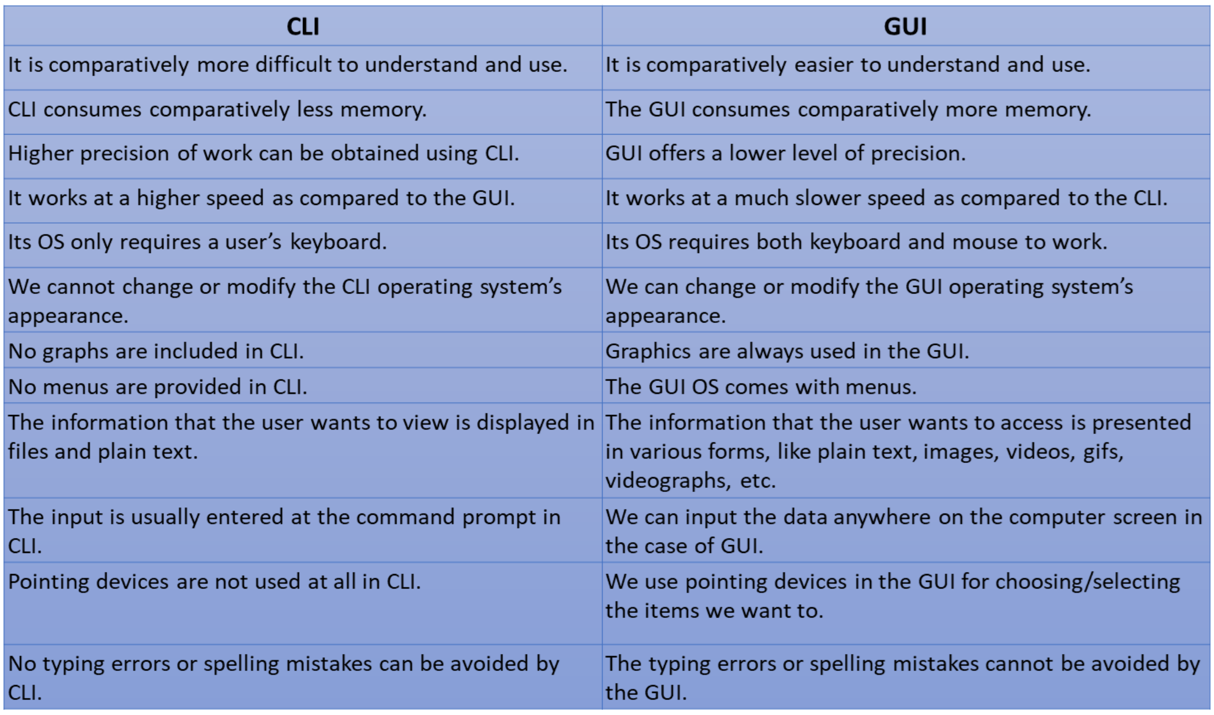
Windows	**Linux**
Windows is an OS within which ASCII text file is inaccessible. it's designed for the people who do not have programming ability and can you the OS as an graphic user interface.	Linux could be a free and open supply OS supported operating system standards. It provides programming interface still as programm compatible with operating system primarily based systems and provides giant selection applications. It is free of cost.
It is costly and needs license. File name is not case sensitive.	File name is case sensitive.
It is less efficient compared to linux. It is less secured.	It is most efficient. It is more secured and used in hacking based systems.

- **Difference between Unix and Linux OS**

Unix	**Linux**

This OS can only be used by copyrighters.	This OS is open source.
This was developed for servers, workstations and mainframes.	ed by anyone either a student or even at the
It supports less file system.	It supports more file system.
It is a complete package of an OS.	It is just a kernel.
It is costly.	It is a free source.

- **Difference between GUI and CLI**



CLI	GUI
It is comparatively more difficult to understand and use.	It is comparatively easier to understand and use.
CLI consumes comparatively less memory.	The GUI consumes comparatively more memory.
Higher precision of work can be obtained using CLI.	GUI offers a lower level of precision.
It works at a higher speed as compared to the GUI.	It works at a much slower speed as compared to the CLI.
Its OS only requires a user's keyboard.	Its OS requires both keyboard and mouse to work.
We cannot change or modify the CLI operating system's appearance.	We can change or modify the GUI operating system's appearance.
No graphs are included in CLI.	Graphics are always used in the GUI.
No menus are provided in CLI.	The GUI OS comes with menus.
The information that the user wants to view is displayed in files and plain text.	The information that the user wants to access is presented in various forms, like plain text, images, videos, gifs, videographs, etc.
The input is usually entered at the command prompt in CLI.	We can input the data anywhere on the computer screen in the case of GUI.
Pointing devices are not used at all in CLI.	We use pointing devices in the GUI for choosing/selecting the items we want to.
No typing errors or spelling mistakes can be avoided by CLI.	The typing errors or spelling mistakes cannot be avoided by the GUI.

- **Introduction to Linux OS**
  - It is a free and open-source operating system and the source code can be modified and distributed to anyone commercially or noncommercially under the GNU General Public License.
  - Initially, Linux was created for personal computers and gradually it was used in other machines like servers, mainframe computers, supercomputers, etc. Nowadays, Linux is also used in embedded systems like routers, automation controls, televisions, digital video recorders, video game consoles, smartwatches, etc.
  - The biggest success of Linux is Android(operating system) it is based on the Linux kernel that is running on smartphones and tablets. Due to android Linux has the largest installed base of all general-purpose operating systems. Linux is generally packaged in a Linux distribution
- The popular Linux distributions are:
- MX Linux
- Manjaro
- Linux Mint

  - elementary
  - Ubuntu
  - Debian
  - Solus
- **Types of interfaces in OS**
  - **Kernel:** It is a lowest level of software to have interface with computer hardware and its processors. It is a head of OS.
  - **Shell:** It acts as an interface between a user and a kernel.
  - **Applications:** All the activities can be stored into a file or a directory and have it as further usage.
- **Shell:**

  AS shell acts as an interface between user and a kernel we do all the operations in the shell using scripting language.

  There are many scripting languages like Bash, Csh, Tcsh, Tcl, Perl, Python ..etc;
- **Unix commands:**

  We have few commands in Unix to create a file, directory, editing, copying, pasting and many more.

  All the scripting in shell is done in terminal just as we do in command prompt in windows OS.
- Let's list out few commands that are most used in Unix.

a.a.i.1. General syntax of commands:

Command –option <argument>

a.a.i.2. Location of a file or directory:

One should know where our file or directory is located in our system, this could be represented using two methods.

a.a.i.2.a. Absolute path:

This describes the complete path of our file or directory.
/ = This indicates root and every path should always start root.
The first directory on any system will be "HOME" directory.
Eg: /home/user/pd/sta/timing

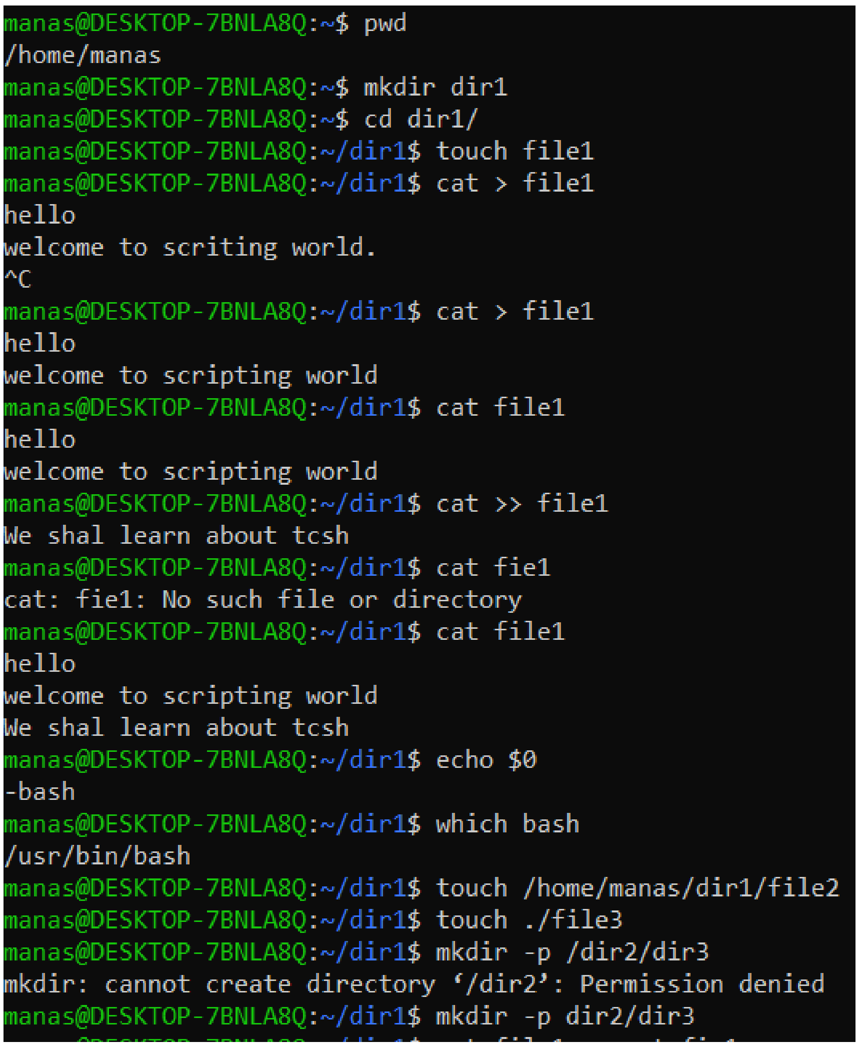
b. Relative path:

This uses few symbols to describe "HOME, CURRENT, PREVIOUS" directories.
./ = This indicates current directory
../ = This indicates previous directory
~ = This indicates home directory
Eg: cd ~ -> Changing from current directory to home directory
Eg: cp ../ ./ -> copying previous directory to current directory

TCSH Commands

1. **echo $0** = Prints which shell we are working in.
2. **which<command/shell>** = Prints the absolute path of the installed location of the shell or the command.
3. **To install any package:** apt install <package name>
   Eg: apt install tcsh
   Sudo install –y <package>
   Eg: sudo install –y tcsh
4. To change from one shell to other shell we just need to type the shell name.
5. Knowing present working directory
   PWD: This command gives the absolute path of the directory we are working in.
6. **Changing to another directory** : cd
   Syntax: cd <path of the directory where we want to change>
   Cd - : it will change to the last opened directory.
7. **Creating an empty file:** touch
   Syntax: touch <filename>
8. **Creating a directory:** mkdir
   Mkdir means make directory; it creates an empty directory.
   Syntax: mkidr<directoryname>
   Mkdir –p directory1/directory2 : This creates a directory2 inside directory1 at one go with the help of option –p (-p: parent).
9. **Concatenate command:**
   The command for concatenate is “**CAT”**
   This command helps in creating a file, displaying a file, writing content into a file, concatenating a file.
   Cat >filename : this command creates a file if it does not exist and prompt blinks in next line expecting to add some information to the file. If a file exists the previously existing data will be overwritten by the new content.
   Cat >>filename : this command creates a file if it does not exist and prompt blinks in next line expecting to add some information to the file. If a file exists the new content will be appended to the existing data.
   Cat filename : this command only displays the content of the file in the terminal.
   Cat –n filename : this command displays the line number and content of the file in the terminal.
10. **Redirectory operators:**
    >, < ,>> these symbols are called as redirectory operators.

- > : outputredirectory operator. This displays content of a file but not in the terminal the content is redirected to other file and one can view it by opening that file.
  If content does not exist in a file it adds but If data is already present it will overwrite.
  Syntax : command > command/filename
  Eg: cat > filename
  Eg: echo “ hello” > file
- >> : output redirectory operator. It works as the previous one but it does not overwrite the content it will append.
  Syntax: command >> argument
  Eg: cat >> filename
- < : input redirectory operator. The output of one command is given as input to the next command.
  Syntax: command <argument>< command <argument>
  Eg: cat file < sort file : this sorts the content of the file and this sorted file is an input to the cat command and displays sorted data in the terminal.



```
manas@DESKTOP-7BNLA8Q:~$ pwd
/home/manas
manas@DESKTOP-7BNLA8Q:~$ mkdir dir1
manas@DESKTOP-7BNLA8Q:~$ cd dir1/
manas@DESKTOP-7BNLA8Q:~/dir1$ touch file1
manas@DESKTOP-7BNLA8Q:~/dir1$ cat > file1
hello
welcome to scriting world.
^C
manas@DESKTOP-7BNLA8Q:~/dir1$ cat > file1
hello
welcome to scripting world
manas@DESKTOP-7BNLA8Q:~/dir1$ cat file1
hello
welcome to scripting world
manas@DESKTOP-7BNLA8Q:~/dir1$ cat >> file1
We shal learn about tcsh
manas@DESKTOP-7BNLA8Q:~/dir1$ cat fie1
cat: fie1: No such file or directory
manas@DESKTOP-7BNLA8Q:~/dir1$ cat file1
hello
welcome to scripting world
We shal learn about tcsh
manas@DESKTOP-7BNLA8Q:~/dir1$ echo $0
-bash
manas@DESKTOP-7BNLA8Q:~/dir1$ which bash
/usr/bin/bash
manas@DESKTOP-7BNLA8Q:~/dir1$ touch /home/manas/dir1/file2
manas@DESKTOP-7BNLA8Q:~/dir1$ touch ./file3
manas@DESKTOP-7BNLA8Q:~/dir1$ mkdir -p /dir2/dir3
mkdir: cannot create directory '/dir2': Permission denied
manas@DESKTOP-7BNLA8Q:~/dir1$ mkdir -p dir2/dir3
```

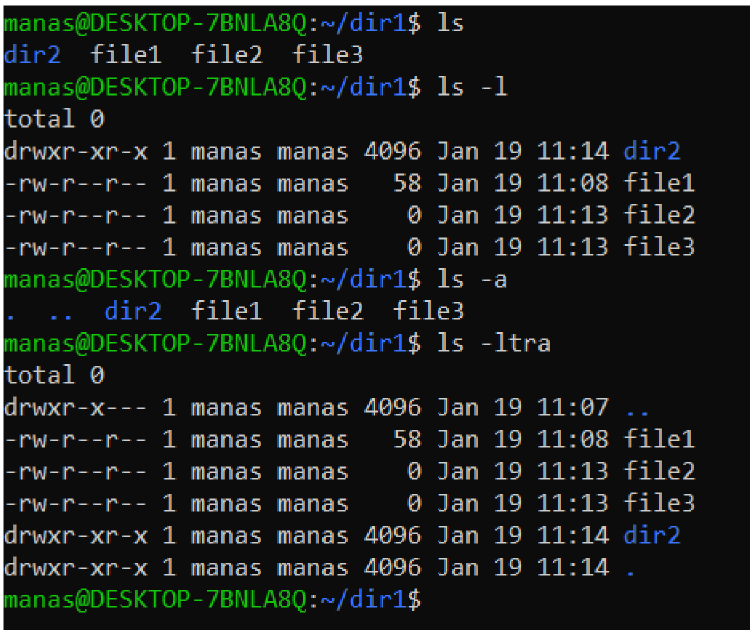
11. **Listing command: ls**

This command lists out all the files and directories in the current directory or if the path is specified it displays the all the directories and files of that particular directory.

Syntax: ls or ls /path of required directory/
Eg :ls ./ or ls ../ orls ~

ls –a : It displays all the hidden files in the directory.
Note: A hidden file is represented with .filename; to create a hidden file : touch .filename
ls –l : It is long listing command which displays as :
-/drw-r--r-- volumeuserID size month date time filename ; - : file ; d : directory
Eg : -rw-r- -r- - 1 userID 4098 jan 1 12:08 filename : This is a file with 1 as a volume by a user with 4098byte created on jan 1 12:08.

Eg: drw-r- -r- - 2 userID 4098 jan 1 12:08 directory : This is a directory with 2 directories or files inside by a user with 4098byte created on jan 1 12:08.
ls –tl : displays long listing output with recently created on the top.
ls –rtl: displays long listing output with recently created on the bottom.
Note: t = time; r = reverse.



```
manas@DESKTOP-7BNLA8Q:~/dir1$ ls
dir2  file1  file2  file3
manas@DESKTOP-7BNLA8Q:~/dir1$ ls -l
total 0
drwxr-xr-x 1 manas manas 4096 Jan 19 11:14 dir2
-rw-r--r-- 1 manas manas   58 Jan 19 11:08 file1
-rw-r--r-- 1 manas manas    0 Jan 19 11:13 file2
-rw-r--r-- 1 manas manas    0 Jan 19 11:13 file3
manas@DESKTOP-7BNLA8Q:~/dir1$ ls -a
.  ..  dir2  file1  file2  file3
manas@DESKTOP-7BNLA8Q:~/dir1$ ls -ltra
total 0
drwxr-x--- 1 manas manas 4096 Jan 19 11:07 ..
-rw-r--r-- 1 manas manas   58 Jan 19 11:08 file1
-rw-r--r-- 1 manas manas    0 Jan 19 11:13 file2
-rw-r--r-- 1 manas manas    0 Jan 19 11:13 file3
drwxr-xr-x 1 manas manas 4096 Jan 19 11:14 dir2
drwxr-xr-x 1 manas manas 4096 Jan 19 11:14 .
manas@DESKTOP-7BNLA8Q:~/dir1$
```

12. **Quotes:**

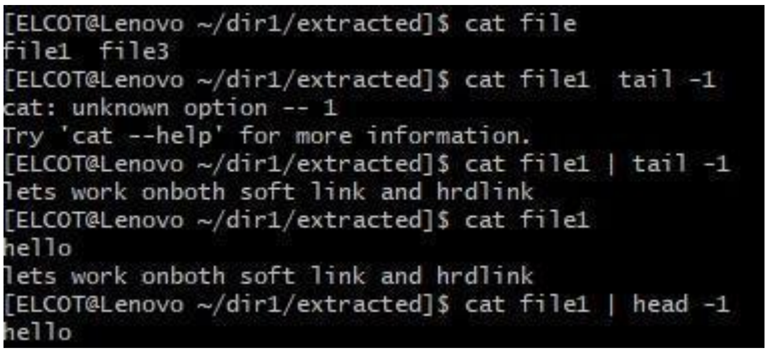
Eg: set a = 10
 Set b = 5
 Echo ‘$a,$b’ : output: 10,5
 Echo “$a,$b” : output: $a,$b
 Set c = `expr $a + $b` : output: 15
 ` `: command substitution
 “ “ : masking
 ‘ ‘ : to access the variables

13. **Tail and Head:** It is used to print last 10 lines and 1st 10 lines of a file respectively.

Tail file: it prints last 10 lines of file
Tail -5 file: it prints last 5 lines of file
Head file: it prints 1st 10 lines of file
Head -4 file: it prints 1st 4 lines of file

14. **Pipeline operator: |**

It is used to cascade the multiple commands.



```
[ELCOT@Lenovo ~/dir1/extracted]$ cat file
file1  file3
[ELCOT@Lenovo ~/dir1/extracted]$ cat file1  tail -1
cat: unknown option -- 1
Try 'cat --help' for more information.
[ELCOT@Lenovo ~/dir1/extracted]$ cat file1 | tail -1
lets work onboth soft link and hrdlink
[ELCOT@Lenovo ~/dir1/extracted]$ cat file1
hello
lets work onboth soft link and hrdlink
[ELCOT@Lenovo ~/dir1/extracted]$ cat file1 | head -1
hello
```

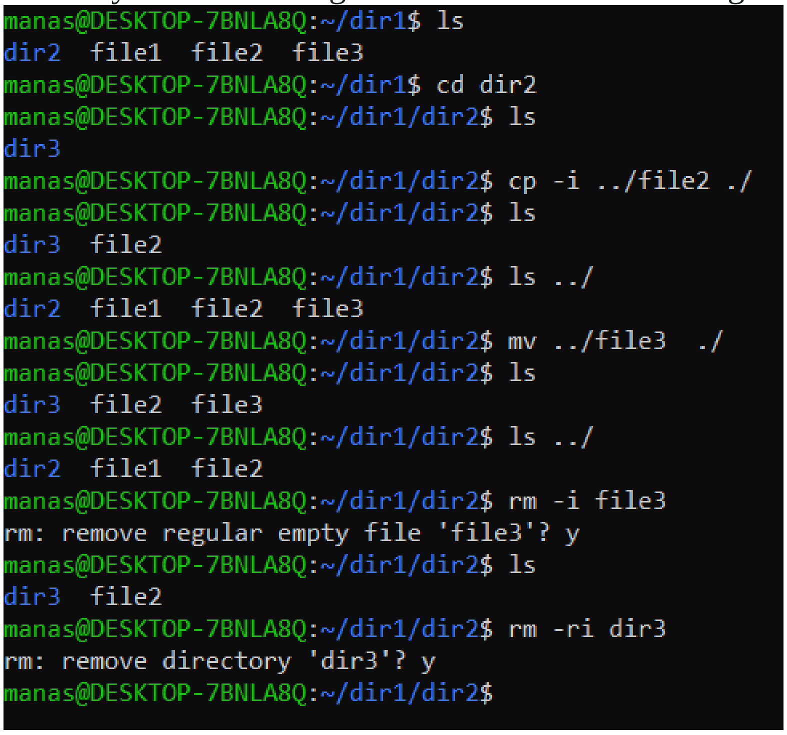
15. **Unix files colors :-**

Blue or Dark Blue – Directory
Green – Executable or recognized data file
Sky Blue – Linked file
Yellow with Black background – Device
Pink – Graphic Image file
Red – Archive file or Zip file

16. **Copying a file or directory :** cp
    Syntax: cp /sourcepath/ /destinationpath/
    We can copy a file or directory using this command. To copy a directory we need to use –r option which means recursive so that all the files and directories inside that directory will be copied. Once a file or directory is copied it will leave the copy behind and gets pasted in the new location also.
    Eg: cp –r /home/user/directory/ ./
    Eg: cp /home/user/directory/file ./

17. **Moving or renaming a file or directory:** mv
    Syntax: mv /sourcepath/ /destinationpath/ : to move
    Syntax: mv /existing file or directory name/ /renaming file or directory name/ : to rename
    This command will delete the copied file or directory in its original location and pastes in the new location. To move a directory we need to use –r option which means recursive so that all the files and directories inside that directory will be moved.
    Eg: mv –r /home/user/directory/ ./
    Eg: mv /home/user/directory/file ./
    Eg: mv /home/user/directory/file1 /home/user/directory/file.txt

18. **Deleting a file or a directory: rm**
    Syntax: rm /path of a file or directory/
    This command will permanently delete the file or directory and we cannot find it in trash bin or anywhere else in the system. To delete a directory we need to use –r option which means recursive so that all the files and directories inside that directory will be delete.
    rm –I file/directory : the prompt asks if we are sure about deleting that particular file or the directory before deleting so that one can avoid wrong deletion.



```
manas@DESKTOP-7BNLA8Q:~/dir1$ ls
dir2  file1  file2  file3
manas@DESKTOP-7BNLA8Q:~/dir1$ cd dir2
manas@DESKTOP-7BNLA8Q:~/dir1/dir2$ ls
dir3
manas@DESKTOP-7BNLA8Q:~/dir1/dir2$ cp -i ../file2 ./
manas@DESKTOP-7BNLA8Q:~/dir1/dir2$ ls
dir3  file2
manas@DESKTOP-7BNLA8Q:~/dir1/dir2$ ls ../
dir2  file1  file2  file3
manas@DESKTOP-7BNLA8Q:~/dir1/dir2$ mv ../file3  ./
manas@DESKTOP-7BNLA8Q:~/dir1/dir2$ ls
dir3  file2  file3
manas@DESKTOP-7BNLA8Q:~/dir1/dir2$ ls ../
dir2  file1  file2
manas@DESKTOP-7BNLA8Q:~/dir1/dir2$ rm -i file3
rm: remove regular empty file 'file3'? y
manas@DESKTOP-7BNLA8Q:~/dir1/dir2$ ls
dir3  file2
manas@DESKTOP-7BNLA8Q:~/dir1/dir2$ rm -ri dir3
rm: remove directory 'dir3'? y
manas@DESKTOP-7BNLA8Q:~/dir1/dir2$
```

19. **Link:** It acts like a string creating pathways for different files and directories in the computer

system. These are capable of creating and storing multiple files in different places referring to one single file.

There are two types of links: soft link, hard link.

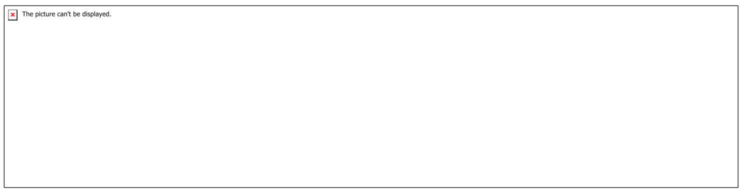
a. **Soft link:** These are the actual copy of the original file. When any changes is done in original file or linked file the changes are affected on both the files. If we delete the original file then the soft linked file will be existing but the content will be erased and the deleted original file will be blinking and soft linked file will appear in red letters in a black background this can be seen when one does ls –l.

   Syntax: ln –s [target file] [soft link file name]
   S indicates its symbiolic i.e. soft link.
   To check if link file is created or not :
   ls –l is the command to check, the output of this command will be:
   lrw-r- -r- - 1 USERID 4098 month date time linkedfile ->originalfile
   The linked file appears in skyblue colour.
   Syntax: ln –sv [target file] [soft link file name]
   V indicates its verbose.
   The output of ln –sv [target file] [soft link file] will be soft link file -> original file ; where soft linked file appears in skyblue colour.

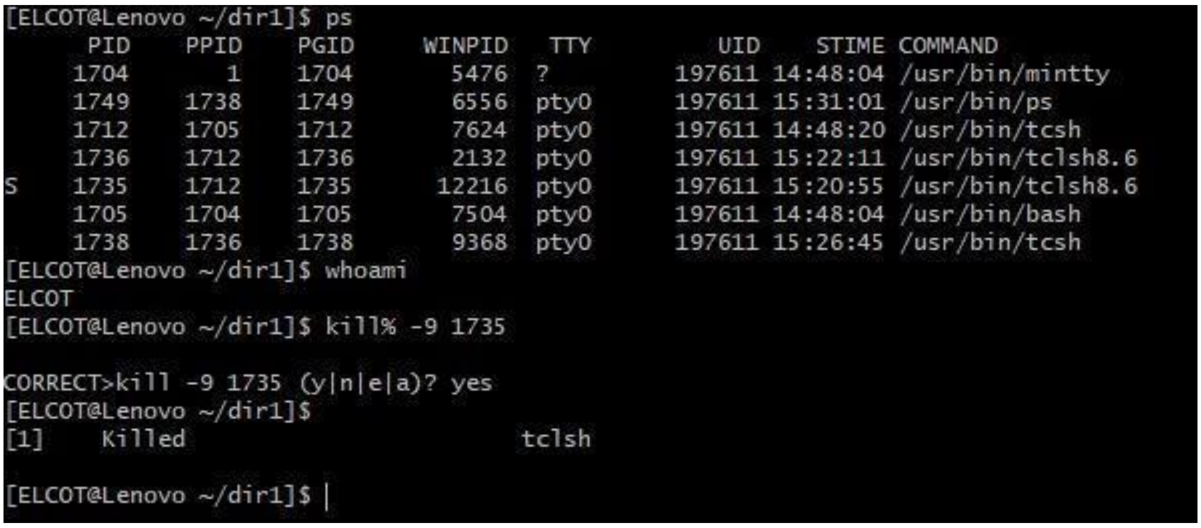
b. **Hard link:** These are the mirror copy of the original file. When any changes is done in original file or linked file the changes are not affected on the other file. If we delete the original file then the hard linked file will be existing and the content will also be existing. The hard link file name appears in blue background.

   Syntax: ln [target file] [hard link file name]
   To check if link file is created or not :
   ls –l is the command to check, the output of this command will be:
   lrw-r- -r- - 1 USERID 4098 month date time linkedfile
   The linked file appears in blue colour background.
   Syntax: ln –v [target file] [hard link file name]
   V indicates its verbose.
   The output of ln –v [target file] [hard link file] will be hard link file => original file ; where hard linked file appears in blue colour background.

20. **Unlink :** This removes the linked file.
    Syntax: unlink <linked file name>

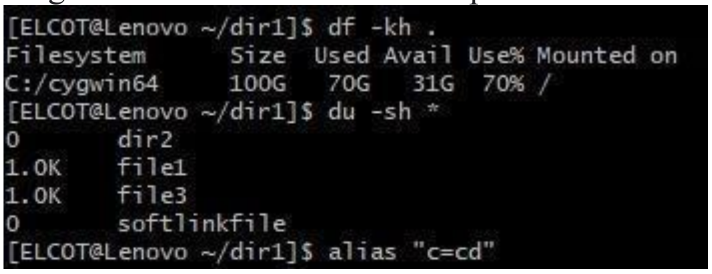


21. **Suspend:** When we are running some script and it takes more than the expected run time then suspension of that script can be done with the help of "ctrl + z". The script still remains in the background so if one wants to resume that script then "bg" will help but it will resume only the recent script that has been suspended.
22. "Command <argument>&" This will work same as ctrl + z and the bg command. So when one gives this command with & we can continue next script instead of waiting for the executing command.
23. **Fg:** foreground command, when one give "fg" command the current running job will showed on the terminal, upon seeing the job one can give bg so that the job still continues in the background.
24. To kill a current process we can use "ctrl + c".
25. **Secure shell :** ssh
    We have to use another server from our server to do our projects and this can be done with help ssh command.
    Syntax :ssh –x <user ID @ server name>
    It asks for the assigned password, once it is entered we are taken to the server where we wish to work.
26. **To know the current running process :** ps (process status).
    The output of ps will be:
    <PID >< terminal type><time><command>
27. **To kill current active process :** kill
    Syntax : kill% -9 PID : it forcefully kills the active PID.
28. **Jobs:** This command is used to list all the jobs that you are running in the background and in the foreground. If prompt returns without any information it means no jobs are active. This command only works on Bash, tcsh, ksh and csh shells only.



```
[ELCOT@Lenovo ~/dir1]$ ps
      PID    PPID    PGID     WINPID   TTY         UID    STIME COMMAND
     1704       1    1704       5476  ?         197611 14:48:04 /usr/bin/mintty
     1749    1738    1749       6556  pty0      197611 15:31:01 /usr/bin/ps
     1712    1705    1712       7624  pty0      197611 14:48:20 /usr/bin/tcsh
     1736    1712    1736       2132  pty0      197611 15:22:11 /usr/bin/tclsh8.6
S    1735    1712    1735      12216  pty0      197611 15:20:55 /usr/bin/tclsh8.6
     1705    1704    1705       7504  pty0      197611 14:48:04 /usr/bin/bash
     1738    1736    1738       9368  pty0      197611 15:26:45 /usr/bin/tcsh
[ELCOT@Lenovo ~/dir1]$ whoami
ELCOT
[ELCOT@Lenovo ~/dir1]$ kill% -9 1735

CORRECT>kill -9 1735 (y|n|e|a)? yes
[ELCOT@Lenovo ~/dir1]$
[1]    Killed                        tclsh

[ELCOT@Lenovo ~/dir1]$ |
```

29. **To know disk information:**
    Staying in the home directory :df –kh ./ : This gives the disk free area of our system or server.
    To get each files disk space utilized information: du –sh *
    To get the information of the complete team's files utilized area information: du –u *

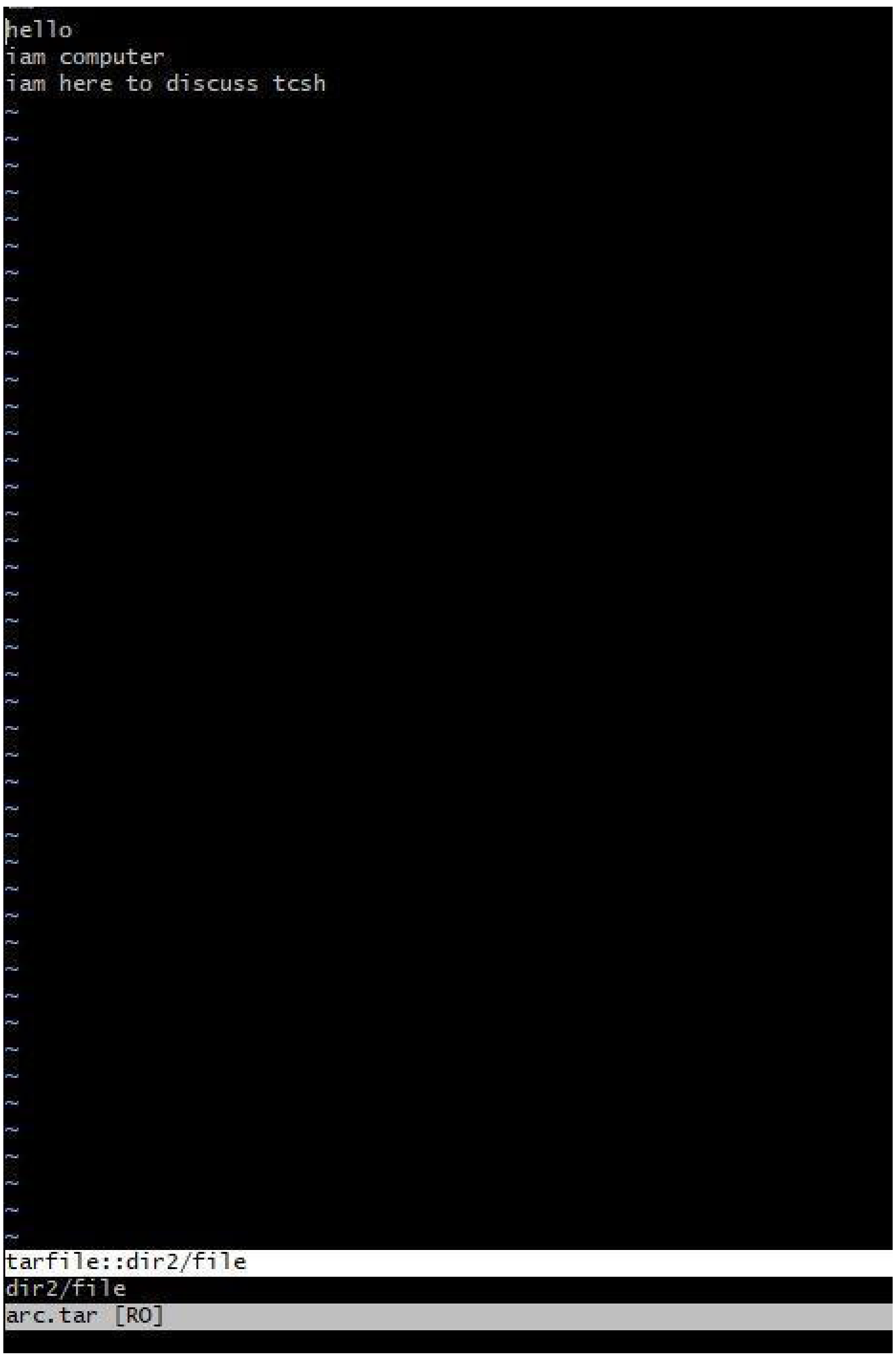


```
[ELCOT@Lenovo ~/dir1]$ df -kh .
Filesystem      Size  Used Avail Use% Mounted on
C:/cygwin64     100G   70G   31G  70% /
[ELCOT@Lenovo ~/dir1]$ du -sh *
0       dir2
1.0K    file1
1.0K    file3
0       softlinkfile
[ELCOT@Lenovo ~/dir1]$ alias "c=cd"
```

30. **To create our own command: alias**
    Syntax: alias "our_command=Linux_command"
    Eg: alias "c=cd"
31. **To clear the terminal :** clear
32. **To send mail from the terminal:** mail
    Syntax: mail –s "subject" <mail id of receiver><message>
    Syntax to send file as a message: mail –s "subject" <mail id of receiver>< filename
    Syntax to add cc, bcc and attach file: mail –s "subject" <mail id of receiver> -c <cc mail id> -b < bcc mail id> -a <attachment><message>
    Syntax for multiple recipients: mail –s "subject" <mail ID 1 , mail ID2, …><message>
33. **Archive: tar**
    -cf : to create an archieved folder
    -xf : to extract an achieved folder
    -rf: to udate any direcory or fie to archieved folder
    -tf: to display content of the archieved folder

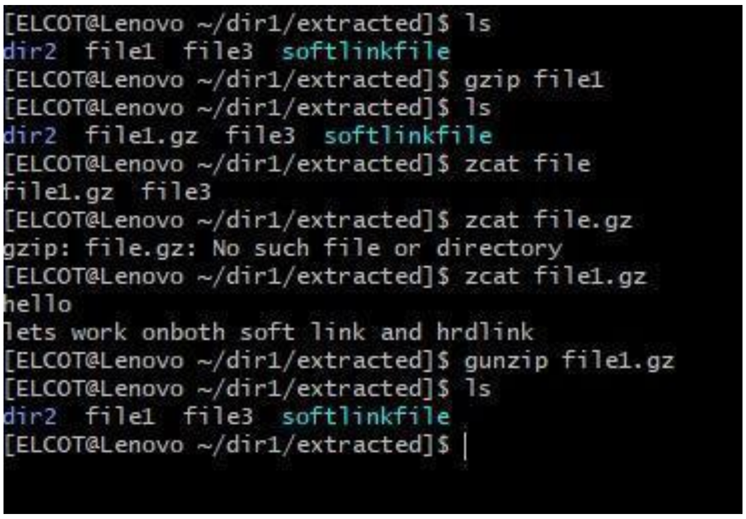
The picture can't be displayed.

The picture can't be displayed.



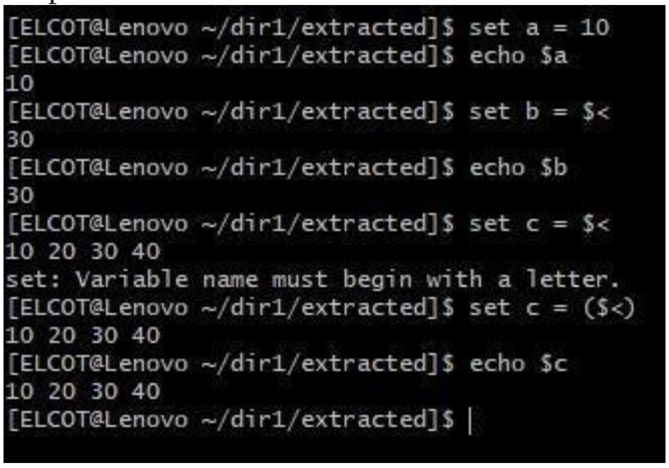
```
hello
iam computer
iam here to discuss tcsh
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
tarfile::dir2/file
dir2/file
arc.tar [RO]
```

34. **To print all the groups that we have access to :** id
    The output of "id" command will be the accessed group name and their ID.
35. **To get access to any of the existing group: mount**
    Mount <active group name>
    Unless the group member gives permission one cannot get access to that group.
36. **To display all the groups:** groups
37. **To display all the groups we are working on:** gid
38. **To change our UNIX password:** password
39. **Sleep n :** n is n seconds that we do not want our terminal to work for. Till the n seconds finish the terminal does not accept any other command entered.
40. **Zip files:**
    To compress a file: gzip<filename>
    To unzip: gunzip<filename>
    To open zipped file: zcat<filename>



```
[ELCOT@Lenovo ~/dir1/extracted]$ ls
dir2  file1  file3  softlinkfile
[ELCOT@Lenovo ~/dir1/extracted]$ gzip file1
[ELCOT@Lenovo ~/dir1/extracted]$ ls
dir2  file1.gz  file3  softlinkfile
[ELCOT@Lenovo ~/dir1/extracted]$ zcat file
file1.gz  file3
[ELCOT@Lenovo ~/dir1/extracted]$ zcat file.gz
gzip: file.gz: No such file or directory
[ELCOT@Lenovo ~/dir1/extracted]$ zcat file1.gz
hello
lets work onboth soft link and hrdlink
[ELCOT@Lenovo ~/dir1/extracted]$ gunzip file1.gz
[ELCOT@Lenovo ~/dir1/extracted]$ ls
dir2  file1  file3  softlinkfile
[ELCOT@Lenovo ~/dir1/extracted]$ |
```

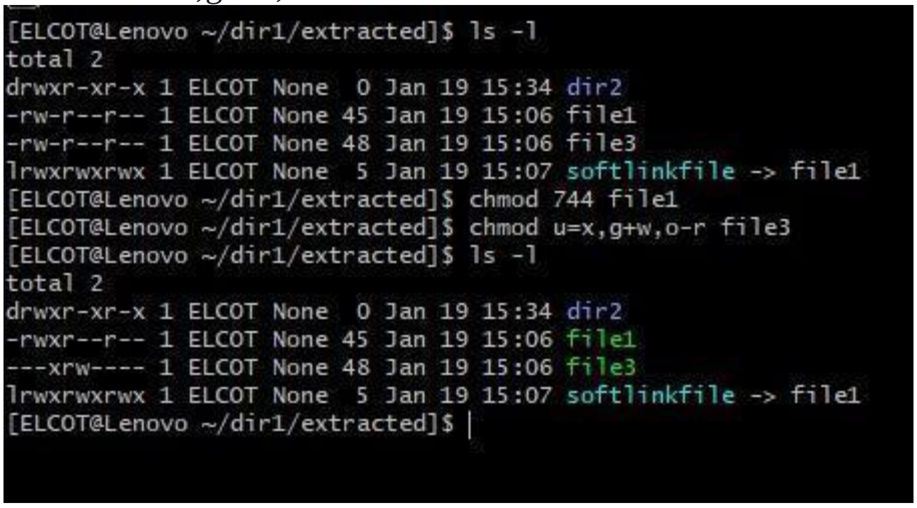
41. **To know who is logged in to your system:** whoami
42. **To assign a variable:** set
    Syntax: set <variable> = <value>
    To print the value of the variable: echo $varaible



```
[ELCOT@Lenovo ~/dir1/extracted]$ set a = 10
[ELCOT@Lenovo ~/dir1/extracted]$ echo $a
10
[ELCOT@Lenovo ~/dir1/extracted]$ set b = $<
30
[ELCOT@Lenovo ~/dir1/extracted]$ echo $b
30
[ELCOT@Lenovo ~/dir1/extracted]$ set c = $<
10 20 30 40
set: Variable name must begin with a letter.
[ELCOT@Lenovo ~/dir1/extracted]$ set c = ($<)
10 20 30 40
[ELCOT@Lenovo ~/dir1/extracted]$ echo $c
10 20 30 40
[ELCOT@Lenovo ~/dir1/extracted]$ |
```

File permissions in unix

When we create a directory or file there are few permissions which are created by

default.
For every file or directory there exists read, write and executable permissions.
There are 3 categories user, group and others to whom we give read, write and executable permissions.
By default the permissions to the directory are: rwx,rx,r .
By default the permissions to the file are: rw,r,r
There are two methods to change the permission to a file or directory: octal method, variable method.
Octal method: It uses 8421 code representation.
Rwx has 8 combinations to be represented.
---: 0
--x: 1
-w-: 2
-wx: 3
r--: 4
r-x: 5
rw-: 6
rwx: 7
For user, group and others this has to be represented.
Variable method: It uses +,-,= to add, remove and assign the permission.
U+w: it adds write option to user
G=wx: it over writes write and executable option to the group
o-x: It removes executable option from others.
The command to change the permissions: chmod
Syntax: chmod<octal/variable notation> file/directory
Eg: I would like to assign user with all permissions, group with read and write permission and others with only read permission using octal method.
Chmod 764 file
Eg: I would like to assign user with all permissions, over writing the write permission to group and others with only read permission by removing write and executable permission using variable method.
Chmodu+rwx,g=rw,o-wx file

```
[ELCOT@Lenovo ~/dir1/extracted]$ ls -l
total 2
drwxr-xr-x 1 ELCOT None  0 Jan 19 15:34 dir2
-rw-r--r-- 1 ELCOT None 45 Jan 19 15:06 file1
-rw-r--r-- 1 ELCOT None 48 Jan 19 15:06 file3
lrwxrwxrwx 1 ELCOT None  5 Jan 19 15:07 softlinkfile -> file1
[ELCOT@Lenovo ~/dir1/extracted]$ chmod 744 file1
[ELCOT@Lenovo ~/dir1/extracted]$ chmod u=x,g+w,o-r file3
[ELCOT@Lenovo ~/dir1/extracted]$ ls -l
total 2
drwxr-xr-x 1 ELCOT None  0 Jan 19 15:34 dir2
-rwxr--r-- 1 ELCOT None 45 Jan 19 15:06 file1
---xrw---- 1 ELCOT None 48 Jan 19 15:06 file3
lrwxrwxrwx 1 ELCOT None  5 Jan 19 15:07 softlinkfile -> file1
[ELCOT@Lenovo ~/dir1/extracted]$ |
```

Editors

The most used editor is gvim. It works just like our word or notepad that we use in windows OS.

Gvim creates file if it doesn't exist. It opens file if it exists.

Syntax: gvim filename

In gvim we have 3 modes of operation:

Command mode
Insert mode
Visual mode

a. Command mode: This mode is main as most of the operations are done in this mode. By default the gvim opens in command mode but if one has to change to command mode from other mode we need to use "esc" key on the keyboard.

Few commands used in gvim are:

- :w – to save
- :q – to quit
- :wq – to save and quit

- :q! – forceful quit , it does not save any changes done.
- :qa – quit all
- :wa – save all
- :vsp – split the gvim in vertical
- :sp – split the gvim in horizontal
- :u – undo
- /searchtext – to search text from top to bottom
- ?searchtext – to search text from bottom to top
- N : move to highlighted text from bottom to top
- n : move to highlighted text from top to bottom
- o : insert line below the cursor
- O : insert line above the cursor
- ^ : to go to start of line
- $: to go to end of line
- gg : to go to start of gvim
- G : to go to end of gvim
- w : move word by word in forward direction
- b : move word by word in backward direction
- %s/searchtext/replacetext/gc : to search and replace a text , if g is not mentioned then only 1st occurrence of each line will be replaced, If g is mentioned then all occurrences will be searched and replaced.
- If c is not mentioned then all the searched text will be replaced at once, if c is mentioned to each searched text before replacing the gvim prompts if we are ok in replacing if yes we have to give y, if no we have to give n.
- /<\searchtext\> : to exclusively search only that particular text
- ggVG: This selects complete gvim file
- h : move cursor left by one character
- j : move cursor down by one character
- k : move cursor up by one character
- l : move cursor right by one character
- :!ls /path/ : to use tcsh shell commands in gvim
- #! /path of installed shell : to run gvim script using any shell scripting

b. Insert mode:

One can edit the content in gvim only in insert mode. We have to change to insert mode from command mode only by using “i” in keyboard.

c. Visual mode:

One can use visual mode to read by highlighting a text, to copy, paste, we can enter into visual mode by pressing “v” key in the keyboard.

Select a text using navigation keys and press “y” in keyboard which means yanking i.e. copying the selected text and enter into command mode using “esc” key and press “p” by keeping the cursor at required place, this pastes the copied content.

One cannot switch between insert mode and visual mode.

File processing commands

1. **Grep: Global Regular Expression**

Grep is a file processing command where one can search for a pattern of a record. When we search for a pattern in a file the whole record containing that particular pattern will be printed on the terminal or can be redirected.

Syntax: grep –option “pattern” filename

Options: -i: Case insensitive.
    -v: Prints the unmatched pattern.
    -E: To match more than one pattern.

Let’s say I have a file with file name as file and the data in it is:

```
Name  id  domain
Kailash 675 PD
Veena768  DV
Keerthi  789 CL
Geetha876  pd
```

Let me search the record kailash using: grep “kailash” file : output ->Kailash 675 PD

Let me search the record pd using grep: grep –i “pd” file : output ->kailash 675 PD
                                                                  Geetha 876 pd

Let me search all the records which donot have DV using grep: grep –v “DV” file

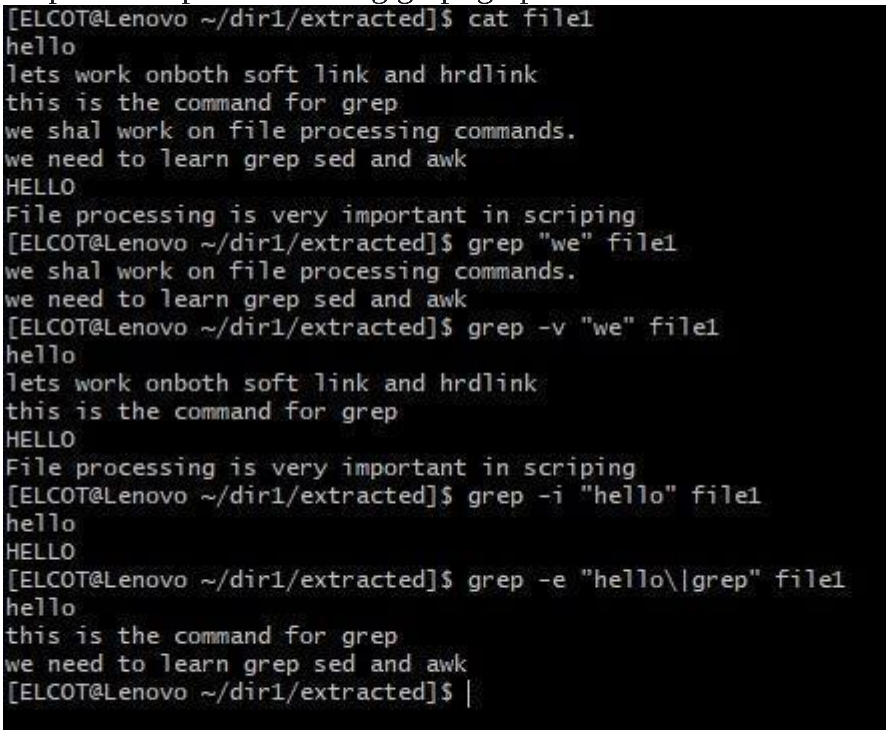
```
Output: Name id domain
Kailash 675 PD
                Keerthi 789 CL
                Geetha 876 pd
```

Let me search all the records with DV and CL: grep –E “DV\|CL” file

Output: Veena 768 DV

Keerthi 789 CL

To print complete file using grep: grep –i "/ / /" file



```
[ELCOT@Lenovo ~/dir1/extracted]$ cat file1
hello
lets work onboth soft link and hrdlink
this is the command for grep
we shal work on file processing commands.
we need to learn grep sed and awk
HELLO
File processing is very important in scriping
[ELCOT@Lenovo ~/dir1/extracted]$ grep "we" file1
we shal work on file processing commands.
we need to learn grep sed and awk
[ELCOT@Lenovo ~/dir1/extracted]$ grep -v "we" file1
hello
lets work onboth soft link and hrdlink
this is the command for grep
HELLO
File processing is very important in scriping
[ELCOT@Lenovo ~/dir1/extracted]$ grep -i "hello" file1
hello
HELLO
[ELCOT@Lenovo ~/dir1/extracted]$ grep -e "hello\|grep" file1
hello
this is the command for grep
we need to learn grep sed and awk
[ELCOT@Lenovo ~/dir1/extracted]$ |
```

2. **Stream editor: sed**

Sed command is used to search, replace, modify, delete, insert, append to a file without altering the content of the original file. It only prints the output on the terminal but does not alter the original file.

Flags used in sed:

r: to read another file
a: to append
s: to search
g: global search
A: line number or number of occurrence
i: to insert
d: to delete
c: to modify
w: to write
p: to print

options used in sed are:

-n : to suppress the line
-f: to read another file
-e: to search multiple patterns

Let's discuss about each flag and options that can be used with those flags in detail:

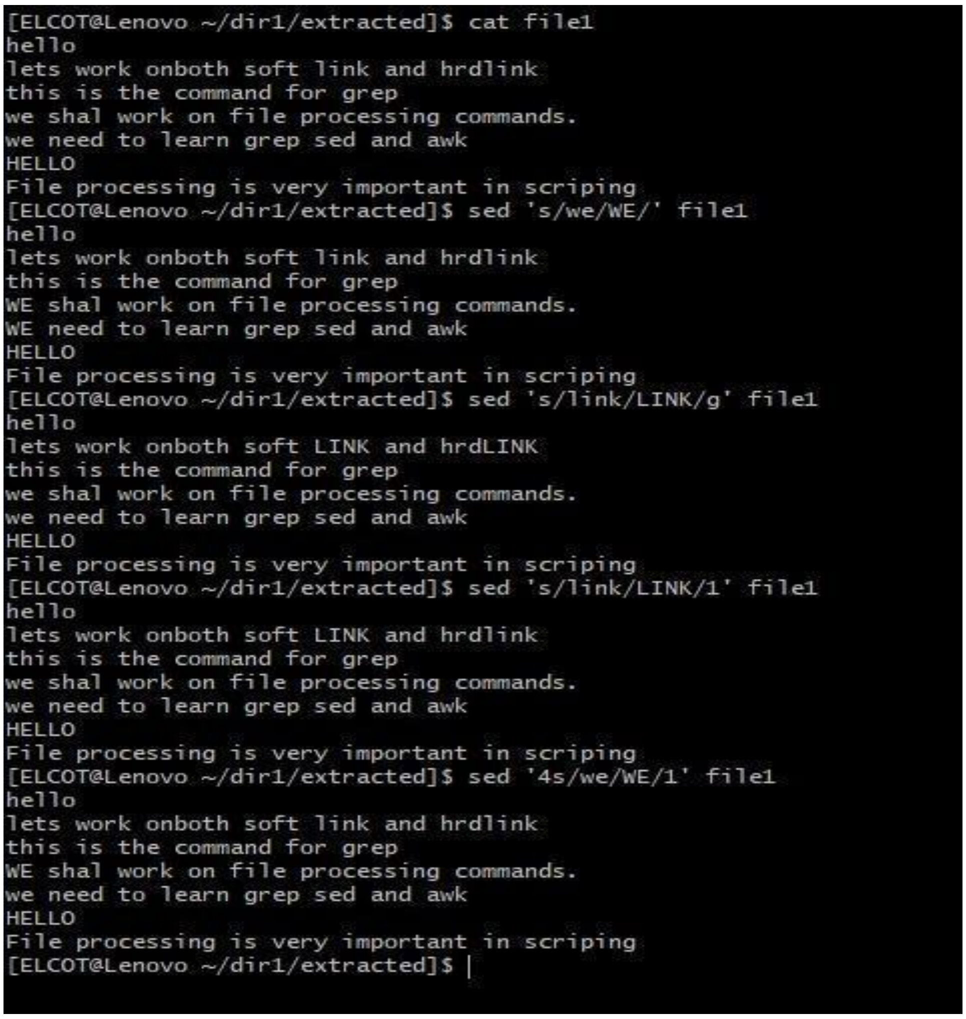
❖ **To search and replace:**
Sed 's/searchtext/replacetext/g'file: It searches and replaces the text globally.

❖ **To search and replace text of particular occurrence of each line.**
Sed 's/search text/replace text/3' file : it searches and replaces the 3 occurrence of each record.

❖ **To search and replace text of particular lines and particular occurrence.**
Sed '2s/search text/replace text/g' file : it searches all the occurrence of a pattern in 2nd record and replace.

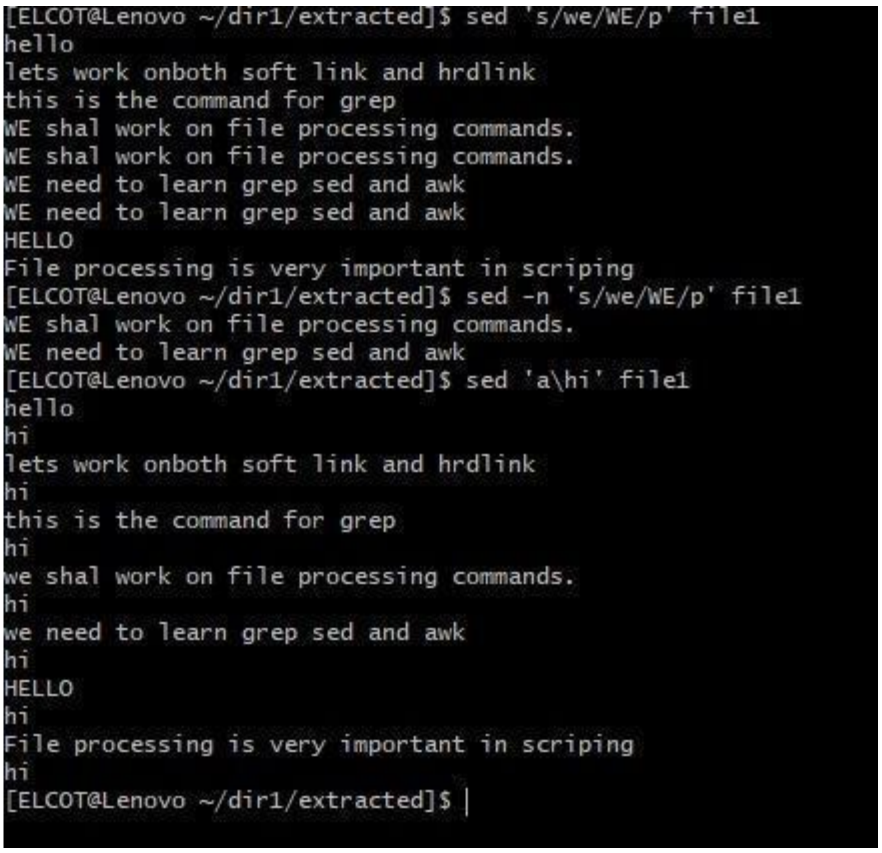
Sed ‘2,5s/search text/replace text/4’ file : it searches and replaces only 4th occurrence of pattern from 2nd to 5th line .
Sed –e ‘2s/search text/replace text/5 ; 5s/search text/replace text/5’ file : it search and replace all 5th occurrences of only 2nd and 5th line.

- **To search and replace using a particular word of a record.**
  Sed ‘/check word/s/search word/replace word/g’ file : It checks for a check word in each line of a file and once it finds that word it then checks if search word is present, If present it will replace it with the replace word If not it will iterate over all the lines of the line.



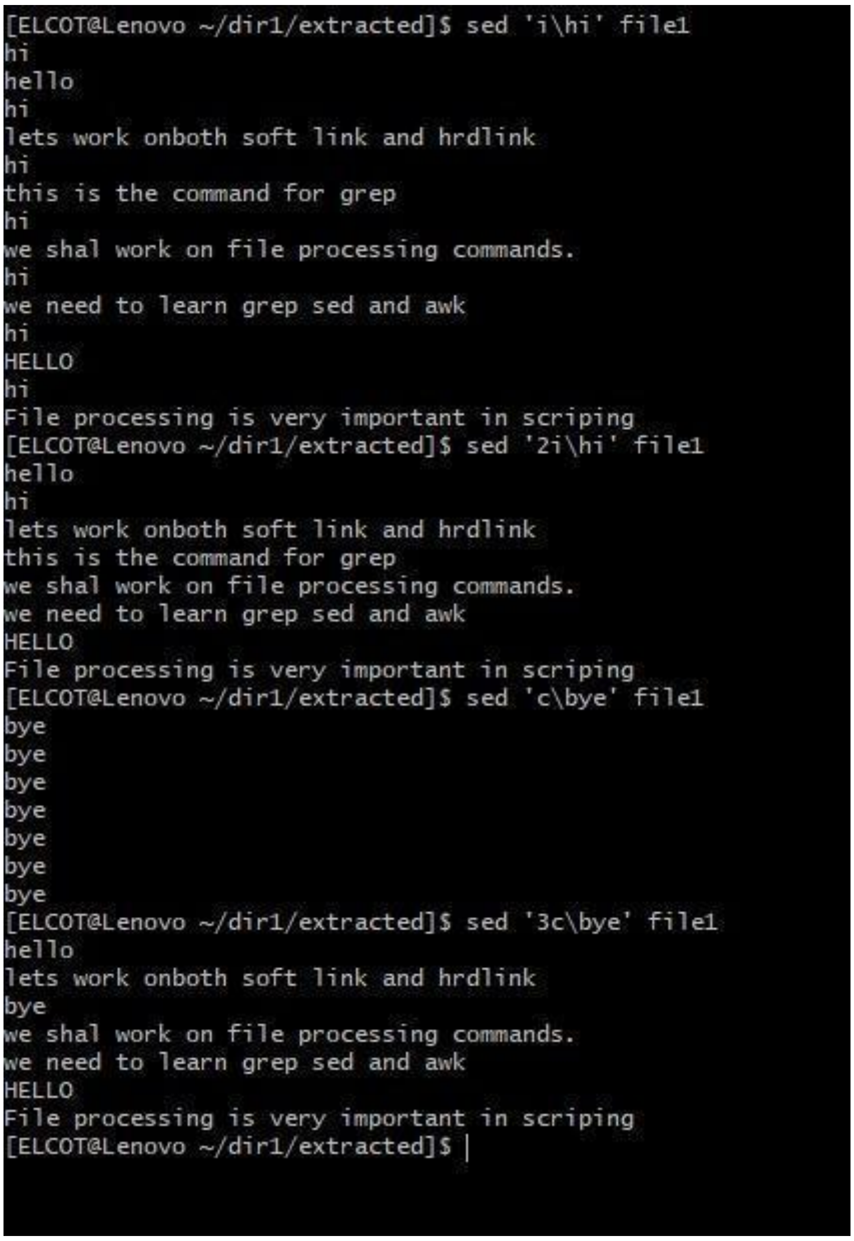
```
[ELCOT@Lenovo ~/dir1/extracted]$ cat file1
hello
lets work onboth soft link and hrdlink
this is the command for grep
we shal work on file processing commands.
we need to learn grep sed and awk
HELLO
File processing is very important in scriping
[ELCOT@Lenovo ~/dir1/extracted]$ sed 's/we/WE/' file1
hello
lets work onboth soft link and hrdlink
this is the command for grep
WE shal work on file processing commands.
WE need to learn grep sed and awk
HELLO
File processing is very important in scriping
[ELCOT@Lenovo ~/dir1/extracted]$ sed 's/link/LINK/g' file1
hello
lets work onboth soft LINK and hrdLINK
this is the command for grep
we shal work on file processing commands.
we need to learn grep sed and awk
HELLO
File processing is very important in scriping
[ELCOT@Lenovo ~/dir1/extracted]$ sed 's/link/LINK/1' file1
hello
lets work onboth soft LINK and hrdlink
this is the command for grep
we shal work on file processing commands.
we need to learn grep sed and awk
HELLO
File processing is very important in scriping
[ELCOT@Lenovo ~/dir1/extracted]$ sed '4s/we/WE/1' file1
hello
lets work onboth soft link and hrdlink
this is the command for grep
WE shal work on file processing commands.
we need to learn grep sed and awk
HELLO
File processing is very important in scriping
[ELCOT@Lenovo ~/dir1/extracted]$ |
```

- **To print replaced text using p flag and –n option:**
  Sed ‘s/search text/replace text/p’ file
  It prints me the original line of file and even the replaced line and as well unmatched line also.
  Sed –n ‘s/search text/replace text/p’ file
  It prints me only the replaced text line of a file.
- **To append a line to a file:**
  Sed ‘a\hi’ file
  It appends hi to all the records of a file.
  Sed ‘2a\hi’ file
  It appends hi to 2nd record of a file.
  Sed ‘2,5a\hi’ file
  It appends hi from 2nd line to 5th line of a file.
  Sed –e ‘2a\hi ; 5a\hi’ file
  It appends hi to 2nd and 5th line of file.



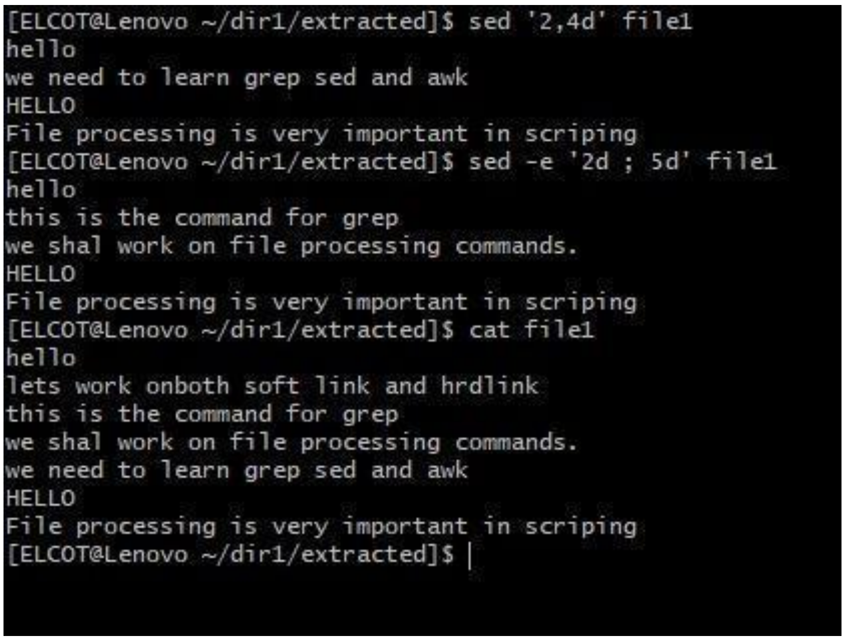
```
[ELCOT@Lenovo ~/dir1/extracted]$ sed 's/we/WE/p' file1
hello
lets work onboth soft link and hrdlink
this is the command for grep
WE shal work on file processing commands.
WE shal work on file processing commands.
WE need to learn grep sed and awk
WE need to learn grep sed and awk
HELLO
File processing is very important in scriping
[ELCOT@Lenovo ~/dir1/extracted]$ sed -n 's/we/WE/p' file1
WE shal work on file processing commands.
WE need to learn grep sed and awk
[ELCOT@Lenovo ~/dir1/extracted]$ sed 'a\hi' file1
hello
hi
lets work onboth soft link and hrdlink
hi
this is the command for grep
hi
we shal work on file processing commands.
hi
we need to learn grep sed and awk
hi
HELLO
hi
File processing is very important in scriping
hi
[ELCOT@Lenovo ~/dir1/extracted]$ |
```

- **To insert a line to a file:**
  Sed ‘i\hi’ file
  It inserts hi to all the records of a file.
  Sed ‘2i\hi’ file
  It inserts hi to 2nd record of a file.
  Sed ‘2,5i\hi’ file
  It inserts hi from 2nd line to 5th line of a file.
  Sed –e ‘2i\hi ; 5i\hi’ file
  It inserts hi to 2nd and 5th line of file.
- **To modify a file:**
  Sed ‘c\hi’ file
  It replaces each record with hi of the file.
  Sed ‘2c\hi’ file
  It replaces 2nd record with hi.
  Sed ‘2,5c\hi’ file
  It replaces 2nd line to 5th line by hi.
  Sed –e ‘2c\hi ; 5c\hi’ file
  It replaces 2nd and 5th line by hi.



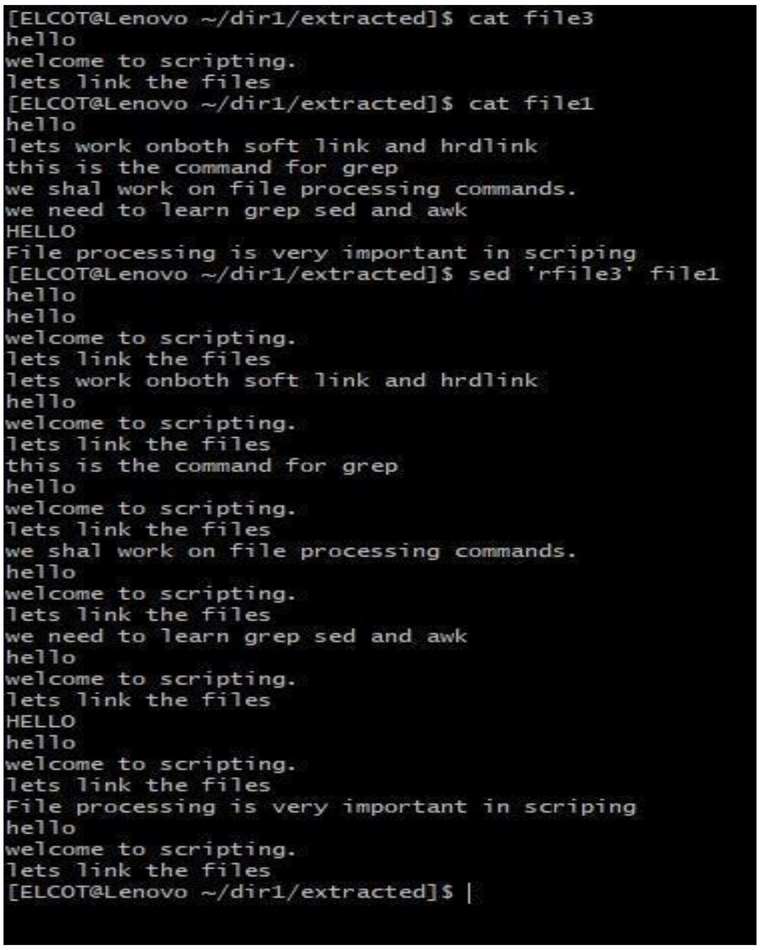
```
[ELCOT@Lenovo ~/dir1/extracted]$ sed 'i\hi' file1
hi
hello
hi
lets work onboth soft link and hrdlink
hi
this is the command for grep
hi
we shal work on file processing commands.
hi
we need to learn grep sed and awk
hi
HELLO
hi
File processing is very important in scriping
[ELCOT@Lenovo ~/dir1/extracted]$ sed '2i\hi' file1
hello
hi
lets work onboth soft link and hrdlink
this is the command for grep
we shal work on file processing commands.
we need to learn grep sed and awk
HELLO
File processing is very important in scriping
[ELCOT@Lenovo ~/dir1/extracted]$ sed 'c\bye' file1
bye
bye
bye
bye
bye
bye
bye
[ELCOT@Lenovo ~/dir1/extracted]$ sed '3c\bye' file1
hello
lets work onboth soft link and hrdlink
bye
we shal work on file processing commands.
we need to learn grep sed and awk
HELLO
File processing is very important in scriping
[ELCOT@Lenovo ~/dir1/extracted]$ |
```

❖ **To delete a line of a file.**
Sed ‘d’ file
It deletes to all the records of a file.
Sed ‘2di’ file
It deletes 2nd record of a file.
Sed ‘2,5d’ file
It deletes from 2nd line to 5th line of a file.
Sed –e ‘2d ; 5d’ file
It deletes 2nd and 5th line of file.



```
[ELCOT@Lenovo ~/dir1/extracted]$ sed '2,4d' file1
hello
we need to learn grep sed and awk
HELLO
File processing is very important in scriping
[ELCOT@Lenovo ~/dir1/extracted]$ sed -e '2d ; 5d' file1
hello
this is the command for grep
we shal work on file processing commands.
HELLO
File processing is very important in scriping
[ELCOT@Lenovo ~/dir1/extracted]$ cat file1
hello
lets work onboth soft link and hrdlink
this is the command for grep
we shal work on file processing commands.
we need to learn grep sed and awk
HELLO
File processing is very important in scriping
[ELCOT@Lenovo ~/dir1/extracted]$ |
```

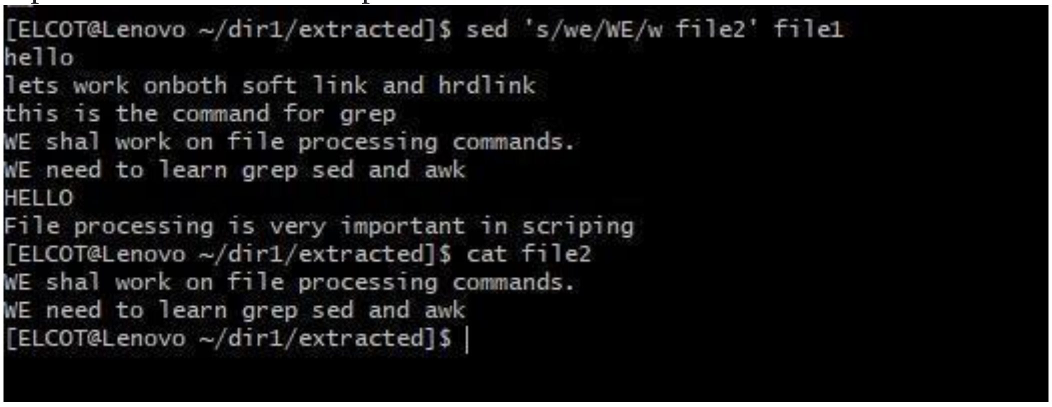
- **To read another file in the current file.**
  Sed ‘rnewfile’ file : It reads new file content below each line of the content of file called file.
  Sed ‘7rnewfile’ file : It reads new file content below 7th line of the file named file.
  Sed ‘/checkword/r newfile’ file : It checks for the check word in each line and if the word is present then the content of new line is printed below the found check word of the file called file.



```
[ELCOT@Lenovo ~/dir1/extracted]$ cat file3
hello
welcome to scripting.
lets link the files
[ELCOT@Lenovo ~/dir1/extracted]$ cat file1
hello
lets work onboth soft link and hrdlink
this is the command for grep
we shal work on file processing commands.
we need to learn grep sed and awk
HELLO
File processing is very important in scriping
[ELCOT@Lenovo ~/dir1/extracted]$ sed 'rfile3' file1
hello
hello
welcome to scripting.
lets link the files
lets work onboth soft link and hrdlink
hello
welcome to scripting.
lets link the files
this is the command for grep
hello
welcome to scripting.
lets link the files
we shal work on file processing commands.
hello
welcome to scripting.
lets link the files
we need to learn grep sed and awk
hello
welcome to scripting.
lets link the files
HELLO
hello
welcome to scripting.
lets link the files
File processing is very important in scriping
hello
welcome to scripting.
lets link the files
[ELCOT@Lenovo ~/dir1/extracted]$ |
```

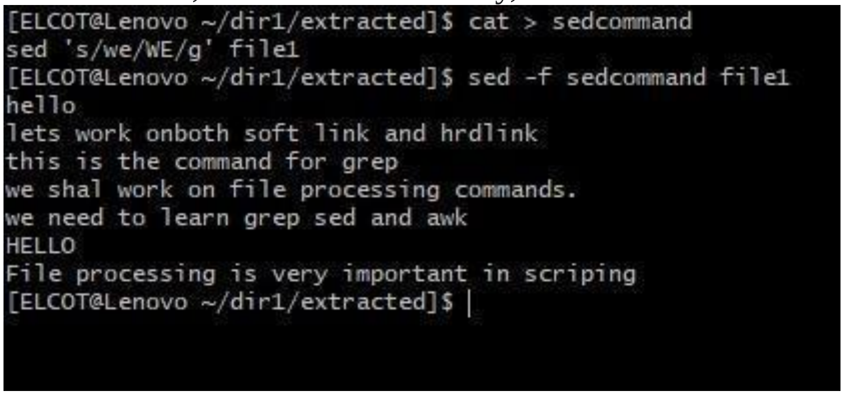
- **To write the sed command output to another file.**
  Sed ‘s/search text/replace text/w file2’ file
  The original lines and the replaced lines are printed as output on the terminal but the

replaced lines are even copied onto file2.



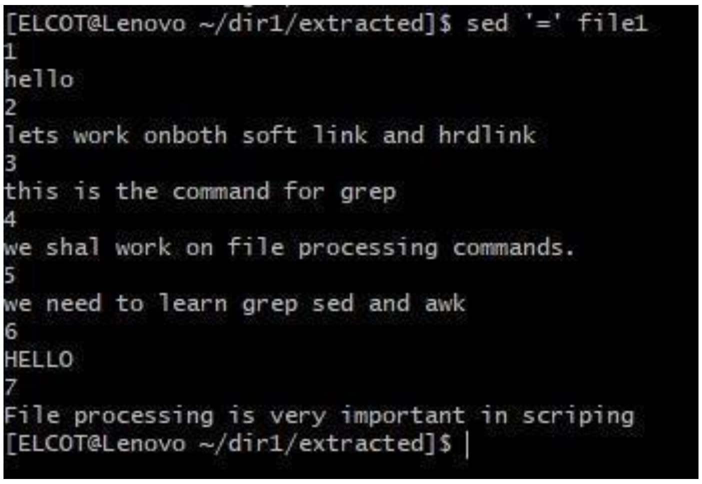
```
[ELCOT@Lenovo ~/dir1/extracted]$ sed 's/we/WE/w file2' file1
hello
lets work onboth soft link and hrdlink
this is the command for grep
WE shal work on file processing commands.
WE need to learn grep sed and awk
HELLO
File processing is very important in scriping
[ELCOT@Lenovo ~/dir1/extracted]$ cat file2
WE shal work on file processing commands.
WE need to learn grep sed and awk
[ELCOT@Lenovo ~/dir1/extracted]$ |
```

- **To use sed command in a file.**
  Sed –f file1 file2
  Lets say file1 has :sed ‘s/this/THIS/g’ file2
  Lets say file2 has: this is India, this is a vast country, this is a secular country.
  Output of sed –f file1 file2 is :
  THIS is India,THIS is a vast country, THIS is a secular country.



```
[ELCOT@Lenovo ~/dir1/extracted]$ cat > sedcommand
sed 's/we/WE/g' file1
[ELCOT@Lenovo ~/dir1/extracted]$ sed -f sedcommand file1
hello
lets work onboth soft link and hrdlink
this is the command for grep
we shal work on file processing commands.
we need to learn grep sed and awk
HELLO
File processing is very important in scriping
[ELCOT@Lenovo ~/dir1/extracted]$ |
```

- **To print line numbers:**
  Sed ‘=’ file



```
[ELCOT@Lenovo ~/dir1/extracted]$ sed '=' file1
1
hello
2
lets work onboth soft link and hrdlink
3
this is the command for grep
4
we shal work on file processing commands.
5
we need to learn grep sed and awk
6
HELLO
7
File processing is very important in scriping
[ELCOT@Lenovo ~/dir1/extracted]$ |
```

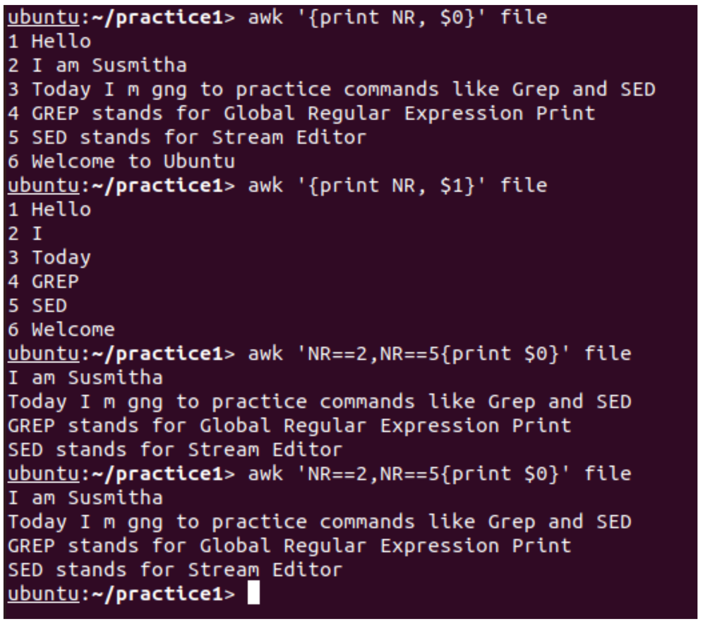
3. **Awk:**
   It is used for pattern scanning and processing, it searches one or more files to see if they contain lines that matches with specified patterns, transforms data files, produce formatted reports, performs arithmetic operations, conditional and loop statements, and then perform the associated actions.

AWK is the abbreviated from the names of the developers – Aho, Weinberger and Kernighan.
Syntax: awk<option> ‘selection criteria{action}’ <input file>

- **Built – in variables of awk:**

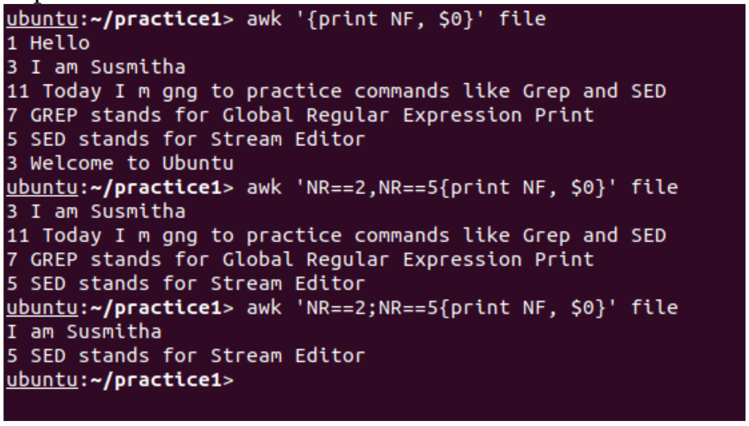
$n : Gvies the fields of records
  Eg: $0 : prints the complete record
    $1: prints the 1st field of the record.

NR: Number of records:
  Prints line numbers of the file.
  Eg: awk ‘{print NR, $0}’ file
  It prints me each line number and the total content.
  Eg: awk ‘NR==2, NR==5{print $0}’ file
  It prints me lines from 2 to 5 including the line numbers and the total content.
  Eg: awk ‘NR==2; NR==5{print $0}’ file
  It prints me only 2nd and 5th line of the line with the line number and the total content.



```
ubuntu:~/practice1> awk '{print NR, $0}' file
1 Hello
2 I am Susmitha
3 Today I m gng to practice commands like Grep and SED
4 GREP stands for Global Regular Expression Print
5 SED stands for Stream Editor
6 Welcome to Ubuntu
ubuntu:~/practice1> awk '{print NR, $1}' file
1 Hello
2 I
3 Today
4 GREP
5 SED
6 Welcome
ubuntu:~/practice1> awk 'NR==2,NR==5{print $0}' file
I am Susmitha
Today I m gng to practice commands like Grep and SED
GREP stands for Global Regular Expression Print
SED stands for Stream Editor
ubuntu:~/practice1> awk 'NR==2,NR==5{print $0}' file
I am Susmitha
Today I m gng to practice commands like Grep and SED
GREP stands for Global Regular Expression Print
SED stands for Stream Editor
ubuntu:~/practice1>
```

NF: Number of fields:
  Prints number of fields of each record.
  Eg: awk ‘{print NF, $0}’ file
  It prints me number of fields of each line and as well the content.
  Eg: awk ‘NR==2, NR==5{print NF, $0}’ file
  It prints me the number of fields and the line content only from 2nd line to 5th line.
  Eg: awk ‘NR==2;NR==5{print NF, $0}’ file
  It prints me the number of fields and the line content of only 2nd and 5th line.



```
ubuntu:~/practice1> awk '{print NF, $0}' file
1 Hello
3 I am Susmitha
11 Today I m gng to practice commands like Grep and SED
7 GREP stands for Global Regular Expression Print
5 SED stands for Stream Editor
3 Welcome to Ubuntu
ubuntu:~/practice1> awk 'NR==2,NR==5{print NF, $0}' file
3 I am Susmitha
11 Today I m gng to practice commands like Grep and SED
7 GREP stands for Global Regular Expression Print
5 SED stands for Stream Editor
ubuntu:~/practice1> awk 'NR==2;NR==5{print NF, $0}' file
I am Susmitha
5 SED stands for Stream Editor
ubuntu:~/practice1>
```

FS: Field separators:

It contains the field separator character which is used divide the record of the file into separators. By default field separator is space.

Eg: awk‘BEGIN{FS=”%”} {print NF}’ file

Let say the file has :

orand%xor

or2%nand nor

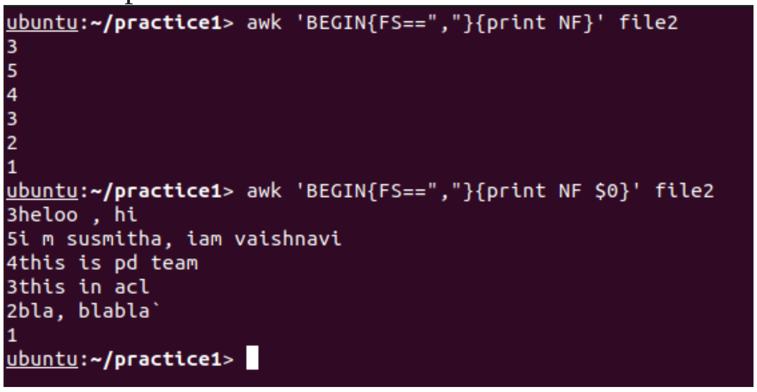
xnor% not %mux

ouput : 2 or and%xor

2 or2%nand nor

3 xnor%not%mux

So FS separates the fields with %.



```
ubuntu:~/practice1> awk 'BEGIN{FS==","}{print NF}' file2
3
5
4
3
2
1
ubuntu:~/practice1> awk 'BEGIN{FS==","}{print NF $0}' file2
3heloo , hi
5i m susmitha, iam vaishnavi
4this is pd team
3this in acl
2bla, blabla`
1
ubuntu:~/practice1> 
```

RS: Record separators:

It has a character which divides the records of file, by default record separators is new line.

Eg: awk‘BEGIN{RS=”%”} {print NF}’ file

Let say the file has :

orand%xor

or2%nand nor

xnor% not %mux

ouput :1 or and

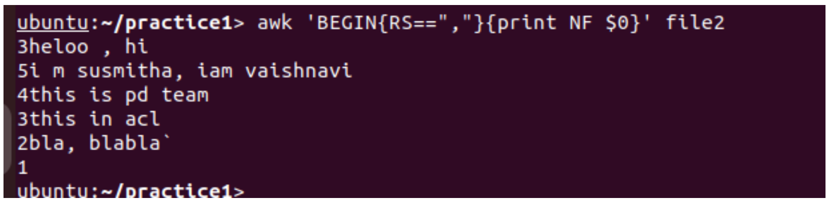
2.xor or2

3. nand nor xnor

4. not

5. mux

So RS separates the records with %.



```
ubuntu:~/practice1> awk 'BEGIN{RS==","}{print NF $0}' file2
3heloo , hi
5i m susmitha, iam vaishnavi
4this is pd team
3this in acl
2bla, blabla`
1
ubuntu:~/practice1>
```

OFS: Output field separators:

It uses a special character to separate the fields of the file and print in terminal as output but by default it is space.

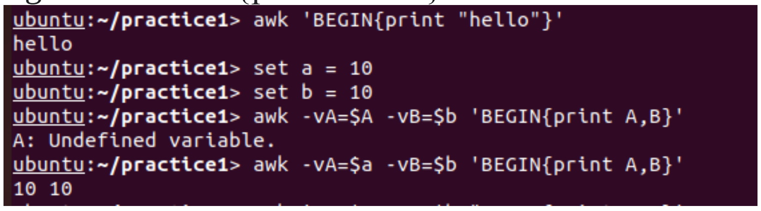
ORS: Output record separators:

It uses a special character to separate the records of the file and print in terminal as output but by default it is new line.

- **Types of processing in awk:**

Pre-processing:

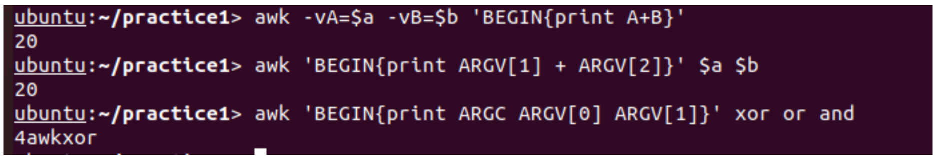
This does not require input file, it uses the keyword “BEGIN” where we can do loops, conditional statements, arithmetic operations using this keyword.
Syntax: awk‘BEGIN{print “text to be printed”}’
Eg: awk‘BEGIN{print “hello”}’



```
ubuntu:~/practice1> awk 'BEGIN{print "hello"}'
hello
ubuntu:~/practice1> set a = 10
ubuntu:~/practice1> set b = 10
ubuntu:~/practice1> awk -vA=$A -vB=$b 'BEGIN{print A,B}'
A: Undefined variable.
ubuntu:~/practice1> awk -vA=$a -vB=$b 'BEGIN{print A,B}'
10 10
```

In awk we have arguments and can be represented using ARGC, ARGV[i]
Eg: awk‘BEGIN{print ARGC}’ and or xor nor
Ouput: 5 ; ARGC is used to print total number of arguments that the awk has and the awk itself is an argument.
Eg: awk‘BEGIN{print ARGV[0], ARGV[1], ARGV[2], ARGC}’ and or
Output: awk, and, or, 3 ;awk is the 0th index, and is 1st index and or is the 2nd index and we have total 3 arguments. So ARGV[i] is used to access the arguments using index.

Mathematical operations:
Awk‘BEGIN{print ARGV[1] * ARGV[2]}’ 10 20
Output: 200
To access external variable using awk:
Eg: set a = 10
      Set b = 20
Awk –vA=$a -vB=$b ‘BEGIN{print A B}’
Output: 10 20
To access external variables using awk and do arithmetic operation:
Eg: awk –vA=$a –vB=$b ‘BEGIN{print A+B}’
Eg: awk ‘{print ARGV[2]/ARGV[1]}’ $a $b
Output: 2
Accessing external variables without using –v:
Eg: awk ‘{print ARGV[1] ARGV[2]}’ $a $b
10 20



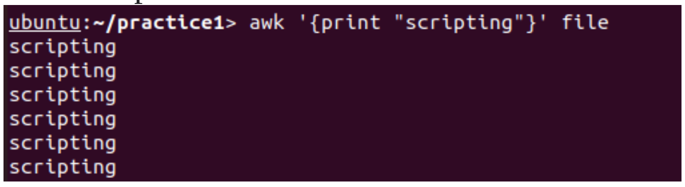
```
ubuntu:~/practice1> awk -vA=$a -vB=$b 'BEGIN{print A+B}'
20
ubuntu:~/practice1> awk 'BEGIN{print ARGV[1] + ARGV[2]}' $a $b
20
ubuntu:~/practice1> awk 'BEGIN{print ARGC ARGV[0] ARGV[1]}' xor or and
4awkxor
```

- **Record – processing:**

It requires input file. It does search operation, it could replace file.
Eg: awk ‘/Hello/{print}’ file
Let say file has:
Hello this is xyz.
Hi this is abc.
Hello world.
Hey dear.

Output: Hello this is xyz.
              Hello world.

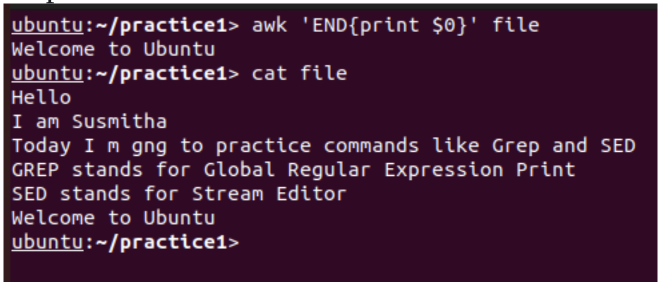
So this prints me only the records which "Hello" in it.
Eg: awk '{print $1 $2}' file
Output: Hello this
Hi this
Hello world
Hey dear.
Eg: awk '{print "scripting"}' file
Output: scripting
Scripting
Scripting
Scripting
So this replaces all the records of the file with scripting.



```
ubuntu:~/practice1> awk '{print "scripting"}' file
scripting
scripting
scripting
scripting
scripting
scripting
```

- **Post-Processing:**

This requires input as file. It used keyword "END" and prints only the last line of file, if NR is used it prints even the last record number.
Eg: awk'END{print NR, $0}' file
Let's say file has:
And or nand
Xor or
Nor not xnor
Ouput: 3 nor not xnor



```
ubuntu:~/practice1> awk 'END{print $0}' file
Welcome to Ubuntu
ubuntu:~/practice1> cat file
Hello
I am Susmitha
Today I m gng to practice commands like Grep and SED
GREP stands for Global Regular Expression Print
SED stands for Stream Editor
Welcome to Ubuntu
ubuntu:~/practice1>
```

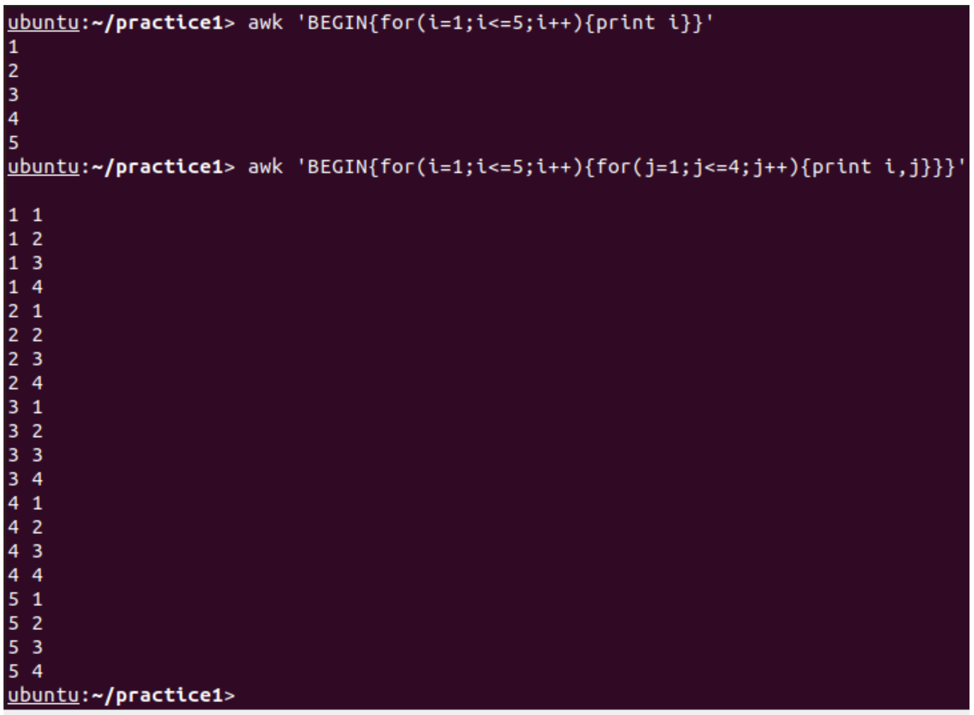
Loops in awk:

- **For loop:**

Syntax: awk'BEGIN{for(initialization;condition;increment){}{print variable}}'

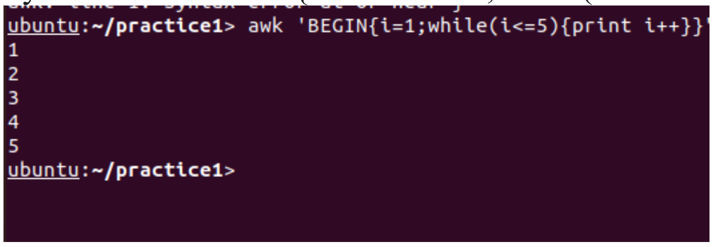
- **Nested loops:**

Syntax:
awk'BEGIN{for(initialization;condition;increment){for(initialization;condition;increment){print variables}}}'


```
ubuntu:~/practice1> awk 'BEGIN{for(i=1;i<=5;i++){print i}}'
1
2
3
4
5
ubuntu:~/practice1> awk 'BEGIN{for(i=1;i<=5;i++){for(j=1;j<=4;j++){print i,j}}}'

1 1
1 2
1 3
1 4
2 1
2 2
2 3
2 4
3 1
3 2
3 3
3 4
4 1
4 2
4 3
4 4
5 1
5 2
5 3
5 4
ubuntu:~/practice1>
```

- **While loops:**

Syntax: awk‘BEGIN{initialization; while(condition) {print increment}}’


```
ubuntu:~/practice1> awk 'BEGIN{i=1;while(i<=5){print i++}}'
1
2
3
4
5
ubuntu:~/practice1>
```

Conditional statements:

- **If:**

Syntax: if (condition) {action}

- **If-else:**

Syntax: if(condition)(action}else{action}

- **If-elseif-else**

Syntax: if(condition){action}elseif (condition) {action } else{action}

Control statements:

- **Break:**

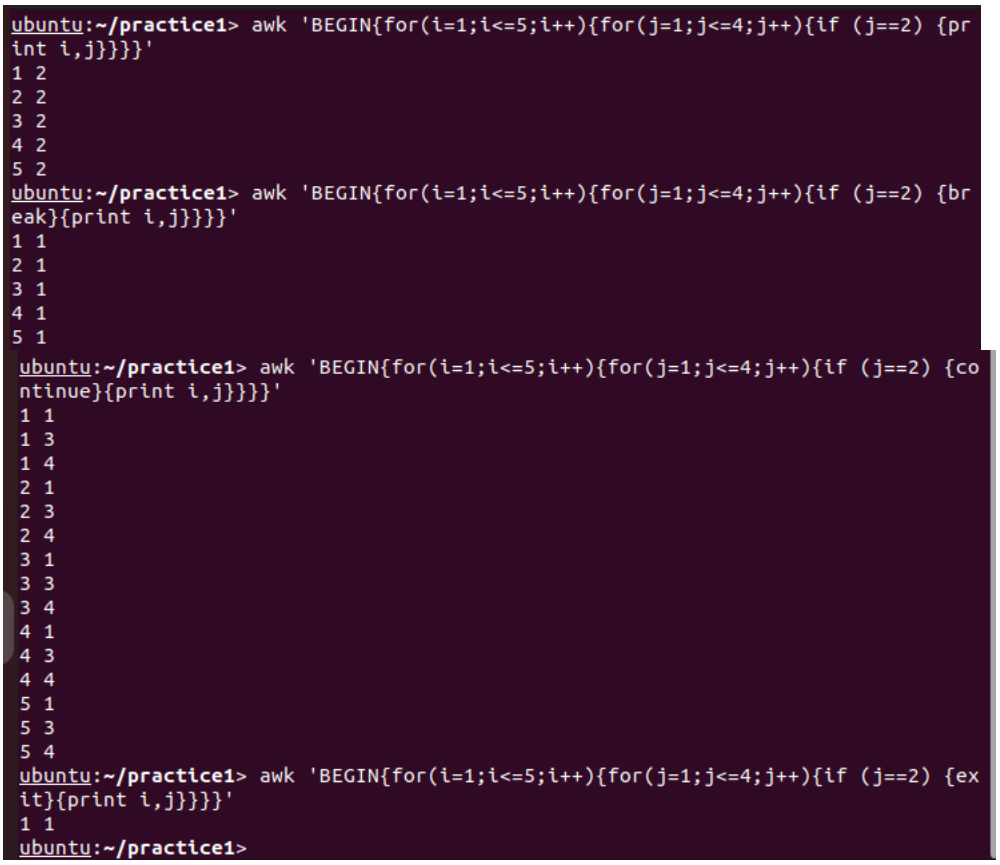
If(condition){break}
Breaks the loop at that condition.

- **Continue:**

If(condition){continue}
Continues the loop skipping that particular condition.

- **Exit:**

If(condition){exit}
Exits the loop at that condition.



```
ubuntu:~/practice1> awk 'BEGIN{for(i=1;i<=5;i++){for(j=1;j<=4;j++){if (j==2) {pr
int i,j}}}}'
1 2
2 2
3 2
4 2
5 2
ubuntu:~/practice1> awk 'BEGIN{for(i=1;i<=5;i++){for(j=1;j<=4;j++){if (j==2) {br
eak}{print i,j}}}}'
1 1
2 1
3 1
4 1
5 1
ubuntu:~/practice1> awk 'BEGIN{for(i=1;i<=5;i++){for(j=1;j<=4;j++){if (j==2) {co
ntinue}{print i,j}}}}'
1 1
1 3
1 4
2 1
2 3
2 4
3 1
3 3
3 4
4 1
4 3
4 4
5 1
5 3
5 4
ubuntu:~/practice1> awk 'BEGIN{for(i=1;i<=5;i++){for(j=1;j<=4;j++){if (j==2) {ex
it}{print i,j}}}}'
1 1
ubuntu:~/practice1>
```

TCSH scripting

We write our tcsh commands in gvim file.
The first line of the file should be the shell interpreter so that the shell understands on which shell it has to execute.
#! / path of the shell installed/
Eg: #! /bin/tcsh
#! -> it is known as shell interpreter.
To execute the file in shell we need to first give the execute permission to the file.
To execute we need to use : shell name /path of the gvim file/
Eg : tcsh ./command.tcsh
If we want some particular commands to not be executed use: so all the commands within if and endif will not be executed.
If 0 then
....
Endif
If we want to execute only few commands and not other commands write “exit” after the commands that are to be executed so that all the commands below exit will not be

executed.

- **Loops:**

Foreach:

```
Foreach variable ($variable)
...
End
```

While:

```
Initialization
While(condition)
Increment
End
```

Conditional statements:

- If(condition) then

```
....
endif
```

- If(condition) then

```
...
Else
...
endif
```

- If(condition) then

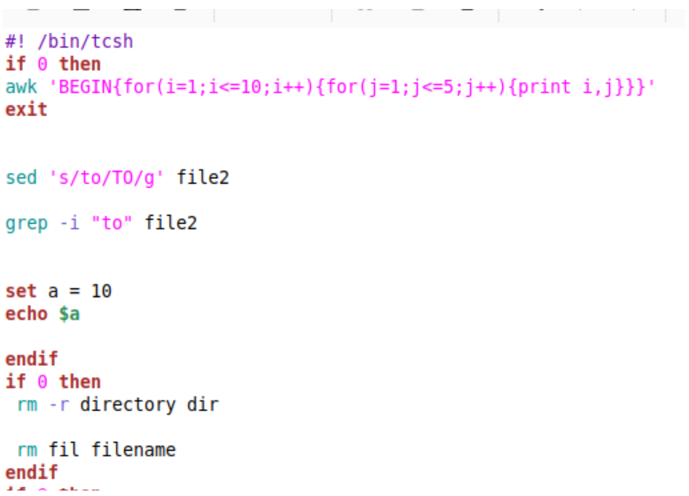
```
....
Elseif(condition) then
...
Else
...
Endif
```

- Switch(strng)

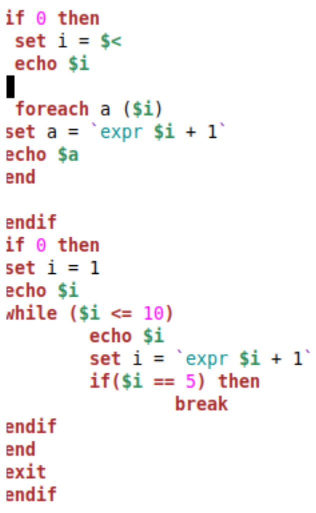
```
Case pattern1:
Breaksw
Case pattern2:
Breaksw
Default:
Breaksw
Endsw
```

Command line arguments:

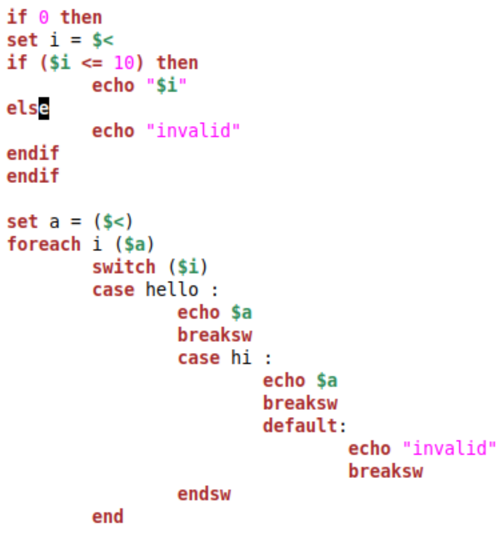
$0: file name or command name
$1,$2... : 1st argument,2nd argument...
$argv[1],$argv[2]...: 1st argument,2nd argument...
$#argv: number of arguments passed while executing.
$* : Special parameter takes entire list as one argument.
$? : Exit status



```
#! /bin/tcsh
if 0 then
awk 'BEGIN{for(i=1;i<=10;i++){for(j=1;j<=5;j++){print i,j}}}'
exit


sed 's/to/TO/g' file2

grep -i "to" file2


set a = 10
echo $a

endif
if 0 then
 rm -r directory dir

 rm fil filename
endif
```

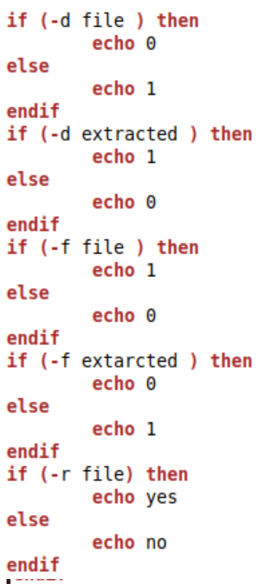


```
if 0 then
 set i = $<
 echo $i

 foreach a ($i)
set a = `expr $i + 1`
echo $a
end

endif
if 0 then
set i = 1
echo $i
while ($i <= 10)
        echo $i
        set i = `expr $i + 1`
        if($i == 5) then
                break
endif
end
exit
endif
```

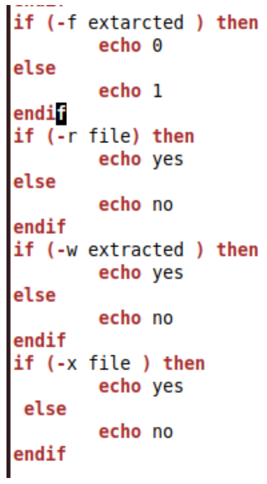


```
if 0 then
set i = $<
if ($i <= 10) then
        echo "$i"
else
        echo "invalid"
endif
endif

set a = ($<)
foreach i ($a)
        switch ($i)
        case hello :
                echo $a
                breaksw
                case hi :
                        echo $a
                        breaksw
                        default:
                                echo "invalid"
                                breaksw
                endsw
        end
```

File operators:
-d: directory
-f : file
-x: executable
-w: writable
-r: readable



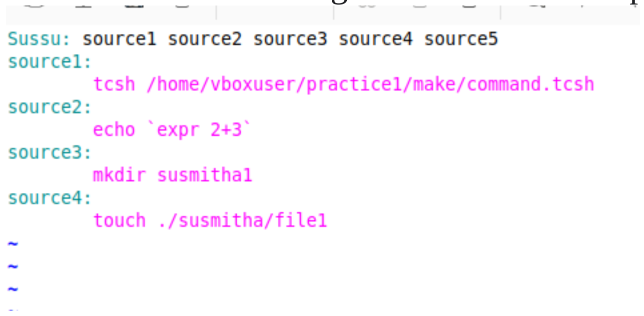
```
if (-d file ) then
        echo 0
else
        echo 1
endif
if (-d extracted ) then
        echo 1
else
        echo 0
endif
if (-f file ) then
        echo 1
else
        echo 0
endif
if (-f extarcted ) then
        echo 0
else
        echo 1
endif
if (-r file) then
        echo yes
else
        echo no
endif
```



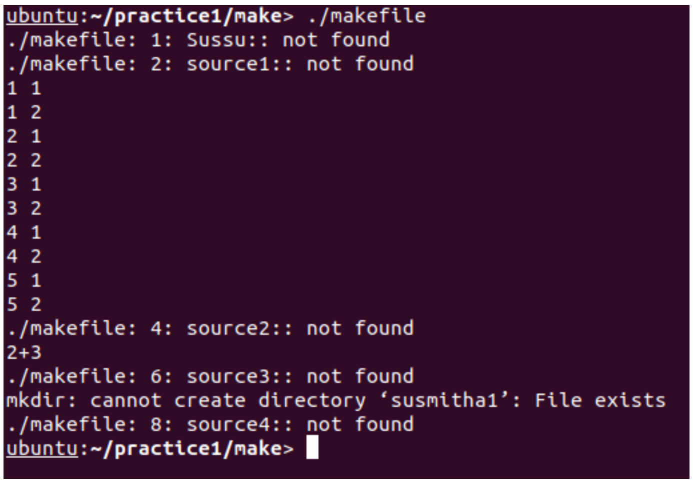
```
if (-f extarcted ) then
        echo 0
else
        echo 1
endif
if (-r file) then
        echo yes
else
        echo no
endif
if (-w extracted ) then
        echo yes
else
        echo no
endif
if (-x file ) then
        echo yes
 else
        echo no
endif
```

Make file

It is used to compile different types of files all together.
The file has to be saved with name “makefile” in a “make” directory.
To execute a makefile :
./makefile
We need to use target and source in the makefile to get access to the file execution.
The makefile has to be given with execution permission.



```
Sussu: source1 source2 source3 source4 source5
source1:
        tcsh /home/vboxuser/practice1/make/command.tcsh
source2:
        echo `expr 2+3`
source3:
        mkdir susmitha1
source4:
        touch ./susmitha/file1
~
~
~
```



```
ubuntu:~/practice1/make> ./makefile
./makefile: 1: Sussu:: not found
./makefile: 2: source1:: not found
1 1
1 2
2 1
2 2
3 1
3 2
4 1
4 2
5 1
5 2
./makefile: 4: source2:: not found
2+3
./makefile: 6: source3:: not found
mkdir: cannot create directory ‘susmitha1’: File exists
./makefile: 8: source4:: not found
ubuntu:~/practice1/make> 
```

TCL scripting

Tcl uses most of the tcsh commands but the syntax varies.

We shall look into tcl commands:

Assigning a value to a variable:

Set a <value>
Eg: set a 10
This prints 10 in the prompt.

To print the value of variable :

Puts <channel ID> $variable
Eg: set a 10
Puts stdout $a
Output: 10

To get input of a variable in terminal:

Gets <stdin> variable
Eg: gets stdin a
%10
Output: 2
It prints me length of characters of a given input.

Ways of assigning a list to variable:

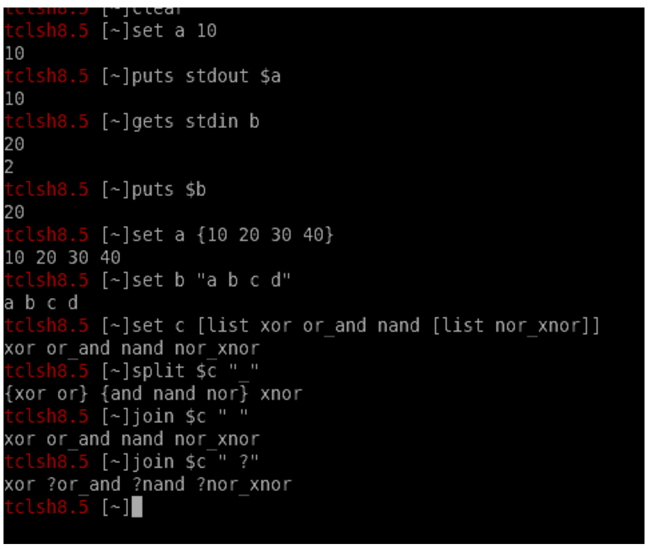
Set a “list of values”
Set a {list of values}
Set a [list <list of values>]
Eg: set a “10 20 30”
Eg: set a {10 20 30}
Eg: set [list 10 20 30]

Command substitution:

Command substitution is used for using a command in another command.
Eg: puts [expr 10 + 20]
Output: 30
The command used for command substitution is: []

Split and join command:

Set a {xor_or and nand_nor xnor_not}
Split $a “_”
Output: {xor} {or and nand} {nor xnor} {not}
Join $a “_”
Output: xor_or_and_nand_nor_xnor_not



```
tclsh8.5 [~]set a 10
10
tclsh8.5 [~]puts stdout $a
10
tclsh8.5 [~]gets stdin b
20
2
tclsh8.5 [~]puts $b
20
tclsh8.5 [~]set a {10 20 30 40}
10 20 30 40
tclsh8.5 [~]set b "a b c d"
a b c d
tclsh8.5 [~]set c [list xor or_and nand [list nor_xnor]]
xor or_and nand nor_xnor
tclsh8.5 [~]split $c "_"
{xor or} {and nand nor} xnor
tclsh8.5 [~]join $c " "
xor or_and nand nor_xnor
tclsh8.5 [~]join $c " ?"
xor ?or_and ?nand ?nor_xnor
tclsh8.5 [~]
```

Mathematical operations:

Expr is used to all the maths operation.
Eg: set a 10
Set b 20
Puts [expr $a + $b]
Output: 30

Expr{abs(x)}

Eg: set a -10
Expr{abs($a)}
Output: 10
It prints me the positive values.

Expr{ceil(x)}

It rounds to highest value of given number.
Eg: set a 10.5
Expr{ceil($a)}
Output: 11

Expr{floor(x)}

It rounds to lowest value of given number.
Eg: set a 10.5
Expr{floor($a)}
Output: 10

Expr{double(x)}

It prints me floating value.

Eg: set a 10
Expr{double($a)}
Output: 10.0

Expr {int(x)}

It prints me the integer value of given number.
Eg: set a 10.5
Expr{int($a)}
Output: 10

Expr{min(range of values)}

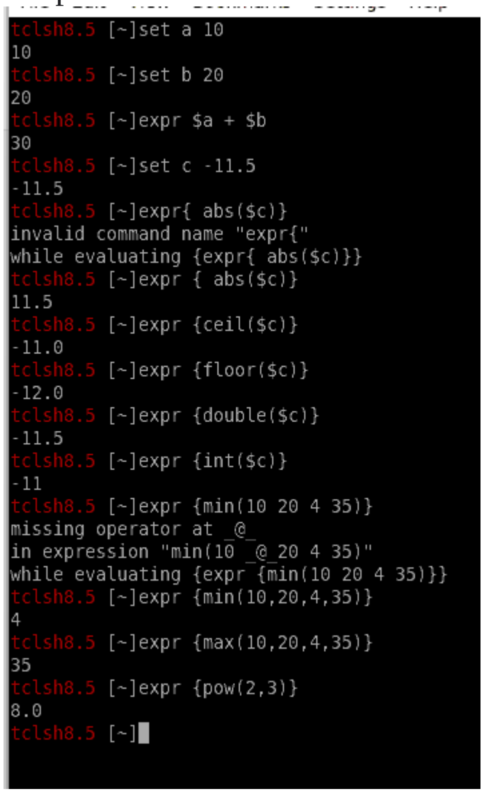
Eg: expr{min(10 30 9 50)}
Output: 9

Expr{max(range of values)}

Eg: expr{max(10 20 3 50)}
Output: 50

Expr{pow(x,y)}

Eg: expr{2,2})
Output: 4



```
tclsh8.5 [~]set a 10
10
tclsh8.5 [~]set b 20
20
tclsh8.5 [~]expr $a + $b
30
tclsh8.5 [~]set c -11.5
-11.5
tclsh8.5 [~]expr{ abs($c)}
invalid command name "expr{"
while evaluating {expr{ abs($c)}}
tclsh8.5 [~]expr { abs($c)}
11.5
tclsh8.5 [~]expr {ceil($c)}
-11.0
tclsh8.5 [~]expr {floor($c)}
-12.0
tclsh8.5 [~]expr {double($c)}
-11.5
tclsh8.5 [~]expr {int($c)}
-11
tclsh8.5 [~]expr {min(10 20 4 35)}
missing operator at _@_
in expression "min(10 _@_20 4 35)"
while evaluating {expr {min(10 20 4 35)}}
tclsh8.5 [~]expr {min(10,20,4,35)}
4
tclsh8.5 [~]expr {max(10,20,4,35)}
35
tclsh8.5 [~]expr {pow(2,3)}
8.0
tclsh8.5 [~]
```

List operations:

We have few list operations where we can modify a list.

Lindex:

Eg: Set a {10 20 30 40}
Lindex $a 0
Output: 10
The index values of a list starts with 0.
We can access last index of a list using “end.
Lindex $a end
Output: 40

Llength:

It prints me the length of a list.
Eg: set a {10 20 30 40}

Llength $a

Output: 4

Lappend:

It appends a value to the list.

It modifies the original list.

Eg: set a {10 20 30 40}

Lappend a xor

Output: 10 20 30 40 xor

Linsert:

It inserts given value at the given index.

Eg: set a {10 20 30 40 50}

Linsert $a 2 and

Output: 10 20 and 30 40 50

Lsort:

It sorts the given list with respect to ascii character, we can you use few switches to sort.

Eg: set a {10 20 AND and 30 AND 40}

Lsort –increasing –unique $a

Output: 10 20 30 40 AND and

-increasing: it sorts in increasing order

-decreasing: it sorts in decreasing order

-unique: it does not print the repeated values in the list.

Lset:

It sets a value to the given index. It modifies the original list.

Eg: set a {10 20 30 40}

Lset a 2 xor

Output: 10 20 xor 40

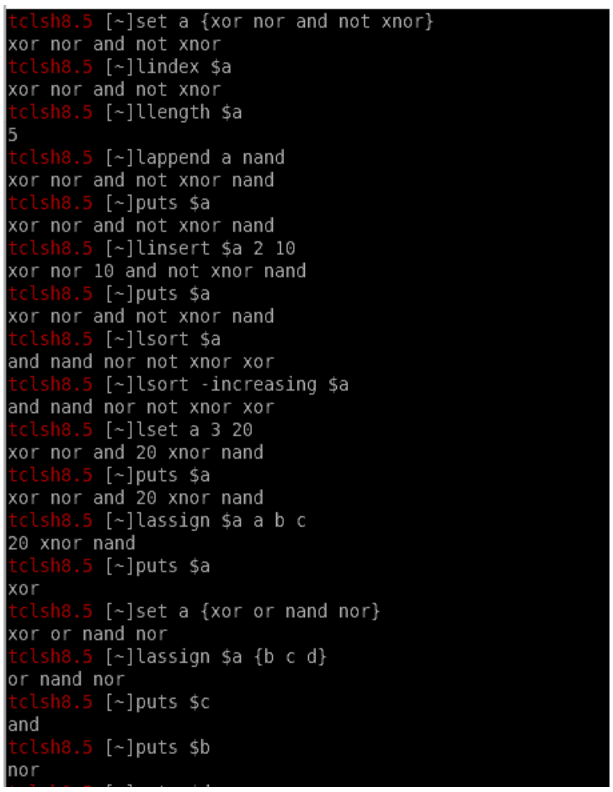
Lassign:

It assigns the values in the list to another variable.

Eg: lassign {10 20 30} a b c

Puts “$a $b $c”

Output: 10 20 30



```
tclsh8.5 [~]set a {xor nor and not xnor}
xor nor and not xnor
tclsh8.5 [~]lindex $a
xor nor and not xnor
tclsh8.5 [~]llength $a
5
tclsh8.5 [~]lappend a nand
xor nor and not xnor nand
tclsh8.5 [~]puts $a
xor nor and not xnor nand
tclsh8.5 [~]linsert $a 2 10
xor nor 10 and not xnor nand
tclsh8.5 [~]puts $a
xor nor and not xnor nand
tclsh8.5 [~]lsort $a
and nand nor not xnor xor
tclsh8.5 [~]lsort -increasing $a
and nand nor not xnor xor
tclsh8.5 [~]lset a 3 20
xor nor and 20 xnor nand
tclsh8.5 [~]puts $a
xor nor and 20 xnor nand
tclsh8.5 [~]lassign $a a b c
20 xnor nand
tclsh8.5 [~]puts $a
xor
tclsh8.5 [~]set a {xor or nand nor}
xor or nand nor
tclsh8.5 [~]lassign $a {b c d}
or nand nor
tclsh8.5 [~]puts $c
and
tclsh8.5 [~]puts $b
nor
```

Lreplace:

It replaces the range of values or a value of the list.

Eg: set a {10 20 30 40 50 60 70}

Lreplace $a 0 0 xor

Output: xor 20 30 40

Lreplace $a 2 4 xor

Output: 10 20 xor 60 70

Lreplace $a 1 1

Output: 10 30 40 50 60 70

Lsearch:

It searches given pattern.

Eg:

Lsearch {a b c}.c

Output: 2

Lsearch –all {a b c bba abb c}.c

Output: 2 5

Lsearch –inline {a b c bba abba c abb}b*

Output: b

Lsearch –inline –all {a b c abba bba bab}b*

Output: b bba bab

Lrange:

It prints range of values.

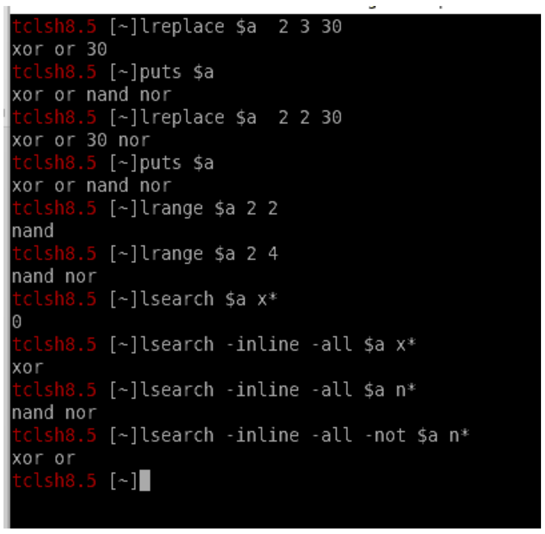
Eg: set a {10 20 30 40 50 60}

Lrange $a 0 0

Output: 10

Lrange $a 0 3

Output: 10 20 30 40



```
tclsh8.5 [~]lreplace $a  2 3 30
xor or 30
tclsh8.5 [~]puts $a
xor or nand nor
tclsh8.5 [~]lreplace $a  2 2 30
xor or 30 nor
tclsh8.5 [~]puts $a
xor or nand nor
tclsh8.5 [~]lrange $a 2 2
nand
tclsh8.5 [~]lrange $a 2 4
nand nor
tclsh8.5 [~]lsearch $a x*
0
tclsh8.5 [~]lsearch -inline -all $a x*
xor
tclsh8.5 [~]lsearch -inline -all $a n*
nand nor
tclsh8.5 [~]lsearch -inline -all -not $a n*
xor or
tclsh8.5 [~]
```

Arrays in tcl:

Arrays are used as single dimension list of values where we can assign it to one variable there by reducing the memory space.

We can assign values to arrays using two methods:

Conventional method:

Set variable(key/index) value

Eg: set a(ones) {1 2 3 4}

Here in the example “a” is my array name, “ones “ is my key of the array where the given list gets stored.

1. **Associative method:**

This is the most used method of assigning values to the array.

There few sets in this method, let’s look at them.

To set values to an array:

Syntax: array set array_name {
Keys {list}
}

Eg: array set numbers {
Ones {1 2 3 4}
Tens {10 20 30 40}
}

To print the array elements:

Syntax: puts stdout $array_name(key)

To print array elements using array:

Syntax: array get array_name : this prints all the keys and their elements.

Syntax: array get array_name key: this prints only the elements of the array of that particular key.

To print size of the array:

Syntax: array size array_name

To print only the elements of the array:

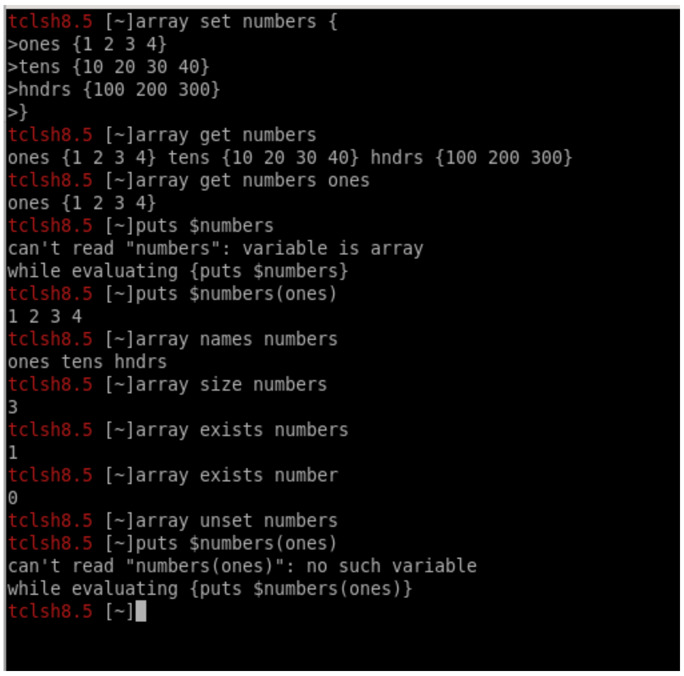
Syntax: array names array_name

To unset the array:

Syntax: array unset array_name

To check if array exists or not:

Syntax: array exists array_name: if yes it prints 1 if no it prints 0.



```
tclsh8.5 [~]array set numbers {
>ones {1 2 3 4}
>tens {10 20 30 40}
>hndrs {100 200 300}
>}
tclsh8.5 [~]array get numbers
ones {1 2 3 4} tens {10 20 30 40} hndrs {100 200 300}
tclsh8.5 [~]array get numbers ones
ones {1 2 3 4}
tclsh8.5 [~]puts $numbers
can't read "numbers": variable is array
while evaluating {puts $numbers}
tclsh8.5 [~]puts $numbers(ones)
1 2 3 4
tclsh8.5 [~]array names numbers
ones tens hndrs
tclsh8.5 [~]array size numbers
3
tclsh8.5 [~]array exists numbers
1
tclsh8.5 [~]array exists number
0
tclsh8.5 [~]array unset numbers
tclsh8.5 [~]puts $numbers(ones)
can't read "numbers(ones)": no such variable
while evaluating {puts $numbers(ones)}
tclsh8.5 [~]
```

Loops in tcl:
For loop:
Syntax: for { initialization } { condition } { increment/decrement } { statement }
Eg: for { set a 1 } { $a <= 5 } { incr a } {puts $a }

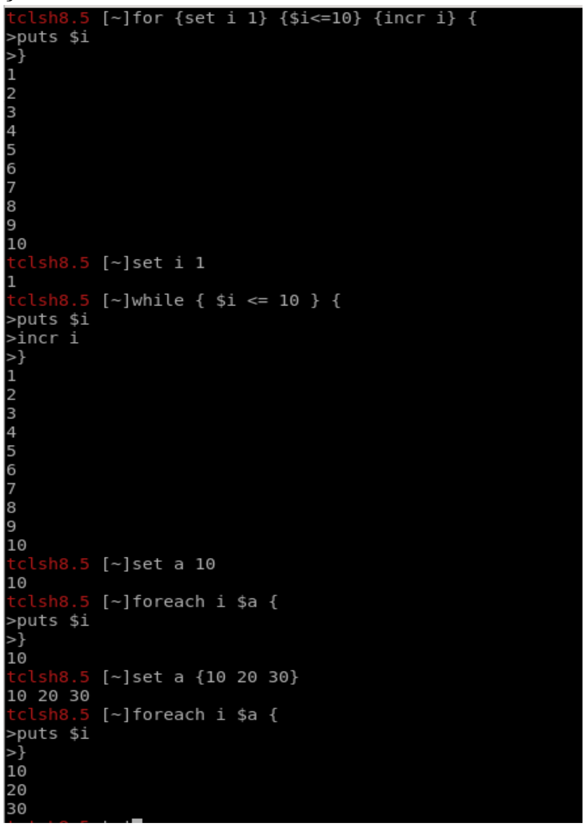
```
Output: 1
        2
        3
        4
        5
```

While loop:
Syntax: initialize while { condition } { statement ; increment/decrement}
Eg: set a 1
while { $a <= 5 } { puts $a incr a}

```
Output: 1
        2
        3
        4
        5
```

Foreach loop:
Syntax: foreach loop_variable variable {statement}
Eg: set a {10 20 30 40}
Foreach i $a {

Puts $i
}



```
tclsh8.5 [~]for {set i 1} {$i<=10} {incr i} {
>puts $i
>}
1
2
3
4
5
6
7
8
9
10
tclsh8.5 [~]set i 1
1
tclsh8.5 [~]while { $i <= 10 } {
>puts $i
>incr i
>}
1
2
3
4
5
6
7
8
9
10
tclsh8.5 [~]set a 10
10
tclsh8.5 [~]foreach i $a {
>puts $i
>}
10
tclsh8.5 [~]set a {10 20 30}
10 20 30
tclsh8.5 [~]foreach i $a {
>puts $i
>}
10
20
30
```

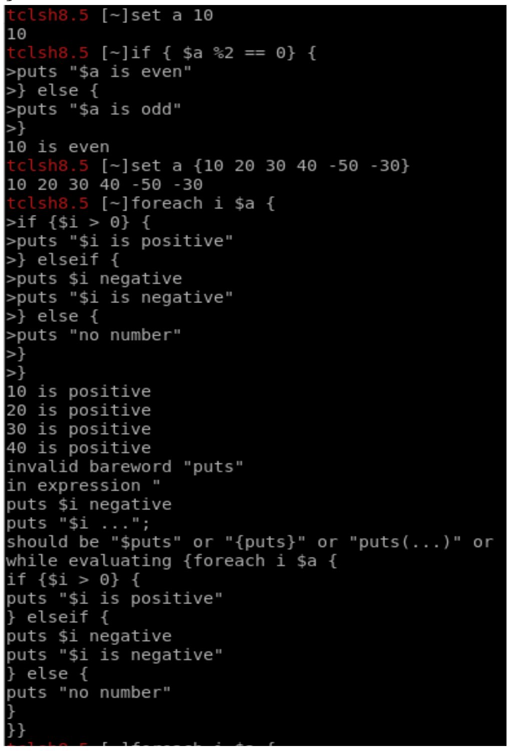
d. Catch: It captures errors. If we run a set of commands from a file and some error has occurred, capturing those errors will be helpful so that we can grep the error and correct the command.

```
  syntax: catch command arguments
  if {[catch {command arg..} result ] } {
  puts stderr $result
  } else {
  puts “no errors”
  }
```

Conditional statements:

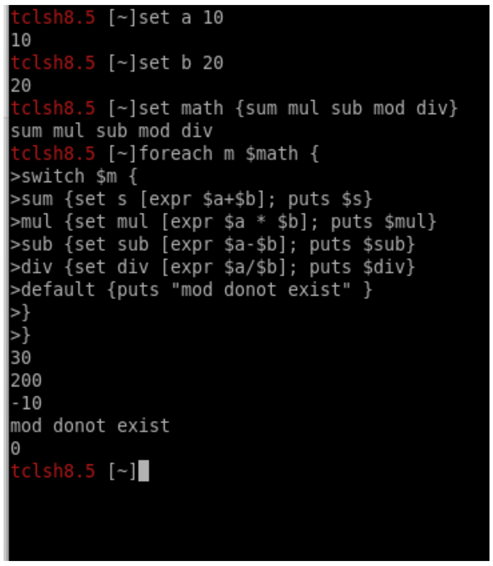
```
If { condition } {
Statement
}
If { condition } {
Statement
}
If { condition } {
Statement
} elseif { condition } {
Statement
} else {
Statement
}
If { condition } {
Statement
} else {
Statement
```

}



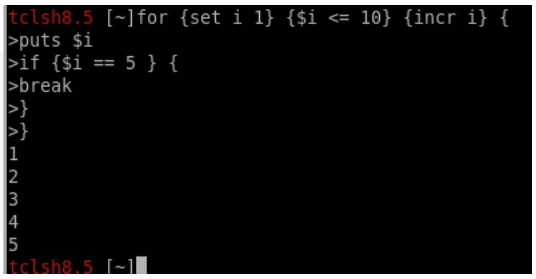
```
tclsh8.5 [~]set a 10
10
tclsh8.5 [~]if { $a %2 == 0} {
>puts "$a is even"
>} else {
>puts "$a is odd"
>}
10 is even
tclsh8.5 [~]set a {10 20 30 40 -50 -30}
10 20 30 40 -50 -30
tclsh8.5 [~]foreach i $a {
>if {$i > 0} {
>puts "$i is positive"
>} elseif {
>puts $i negative
>puts "$i is negative"
>} else {
>puts "no number"
>}
>}
10 is positive
20 is positive
30 is positive
40 is positive
invalid bareword "puts"
in expression "
puts $i negative
puts "$i ...";
should be "$puts" or "{puts}" or "puts(...)" or
while evaluating {foreach i $a {
if {$i > 0} {
puts "$i is positive"
} elseif {
puts $i negative
puts "$i is negative"
} else {
puts "no number"
}
}}
```

Switch string {
Statement 1
}
String 2 {
Statement 2
}
Default {statement}
}
}



```
tclsh8.5 [~]set a 10
10
tclsh8.5 [~]set b 20
20
tclsh8.5 [~]set math {sum mul sub mod div}
sum mul sub mod div
tclsh8.5 [~]foreach m $math {
>switch $m {
>sum {set s [expr $a+$b]; puts $s}
>mul {set mul [expr $a * $b]; puts $mul}
>sub {set sub [expr $a-$b]; puts $sub}
>div {set div [expr $a/$b]; puts $div}
>default {puts "mod donot exist" }
>}
>}
30
200
-10
mod donot exist
0
tclsh8.5 [~]
```

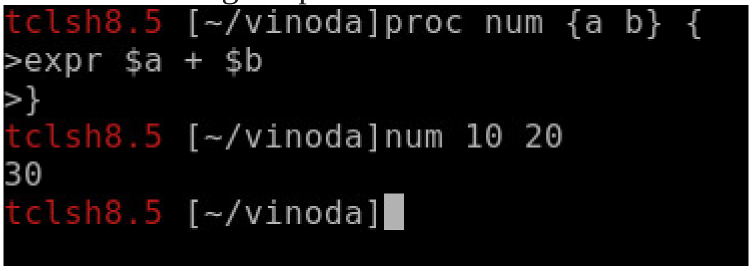
Loop control statements:
Break
Continue
Exit



```
tclsh8.5 [~]for {set i 1} {$i <= 10} {incr i} {
>puts $i
>if {$i == 5 } {
>break
>}
>}
1
2
3
4
5
tclsh8.5 [~]
```

Procedures:
It is a block of code with a series of commands that provide specific reusable functions. This is similar to the functions that we have learnt in C language.
It has two parts: definition and calling.
Syntax: proc procedure_name {arguments} {

```
                Body
                }
               <calling the proc>
```



```
tclsh8.5 [~/vinoda]proc num {a b} {
>expr $a + $b
>}
tclsh8.5 [~/vinoda]num 10 20
30
tclsh8.5 [~/vinoda]
```

Eg: proc hello{}{

```
      Puts "hello world"
}
Hello
```

Output: hello world

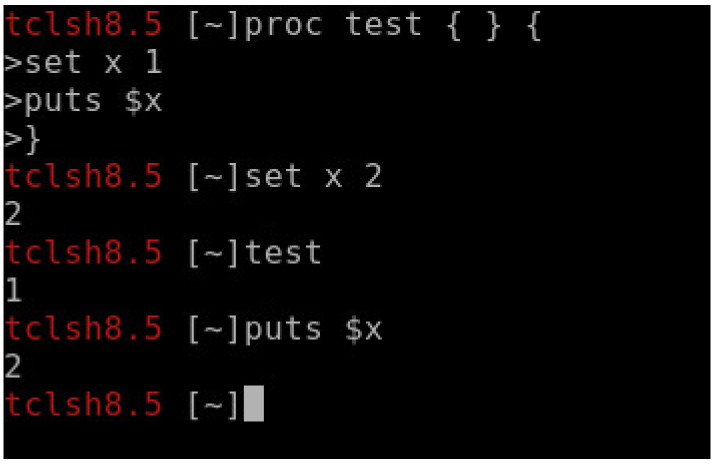
Scope of the procedure:
It has 3 types of scopes: local, global and upvar.
Local scope:
Eg: proc test { } {

```
      Set x 4
      Puts "x is $x"
      }
     Set x 1
      Puts "x is $x"
      Puts {}
     test
      Puts { }
      Puts { }
      Puts "x is $x"
```

Output: x is 1

```
                X is 4
              X is 1
```

Therefore, when x = 4 it is local scope and has to be called inside the definition only.



```
tclsh8.5 [~]proc test { } {
>set x 1
>puts $x
>}
tclsh8.5 [~]set x 2
2
tclsh8.5 [~]test
1
tclsh8.5 [~]puts $x
2
tclsh8.5 [~]
```

Upvar scope:

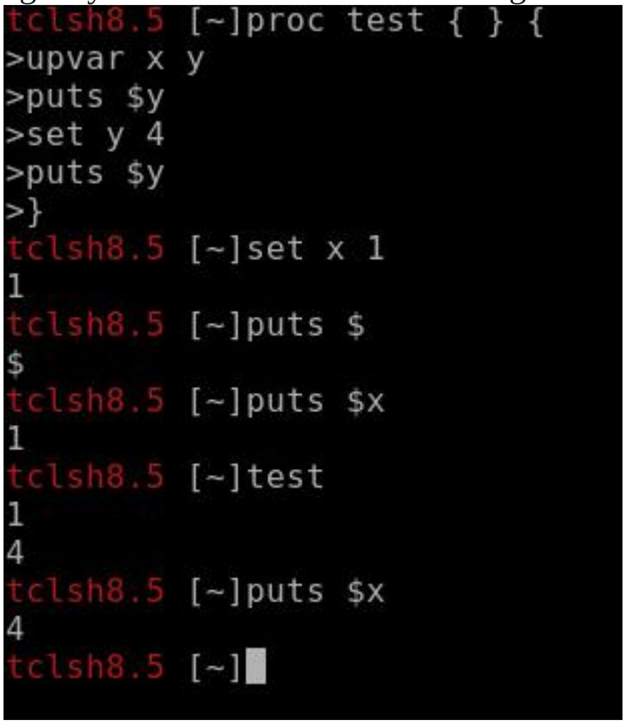
Variable that is outside or one level up.

Syntax: upvar <variable outside> <variable inside procedure>

```
Eg: proc test { } {
      Upvar x y
      Puts "y is $y"
      Set y 4
      Puts "y is $y"
     }
     Set x 1
     Puts " x is $x"
     Test
    Puts "x is $x"
```

```
Output: x is 1
Y is 1
             Y is 4
              X is 4
```

Therefore, initially x is 1 and is printed; but when "test" is called the compilation goes to definition and sees for value of y but it holds the pre-existing value of x into y and only when compilation sees that y is declared with 4 then it sees for printing option and prints again y with value of 4 and this being carried to x and x is declared with value 4.



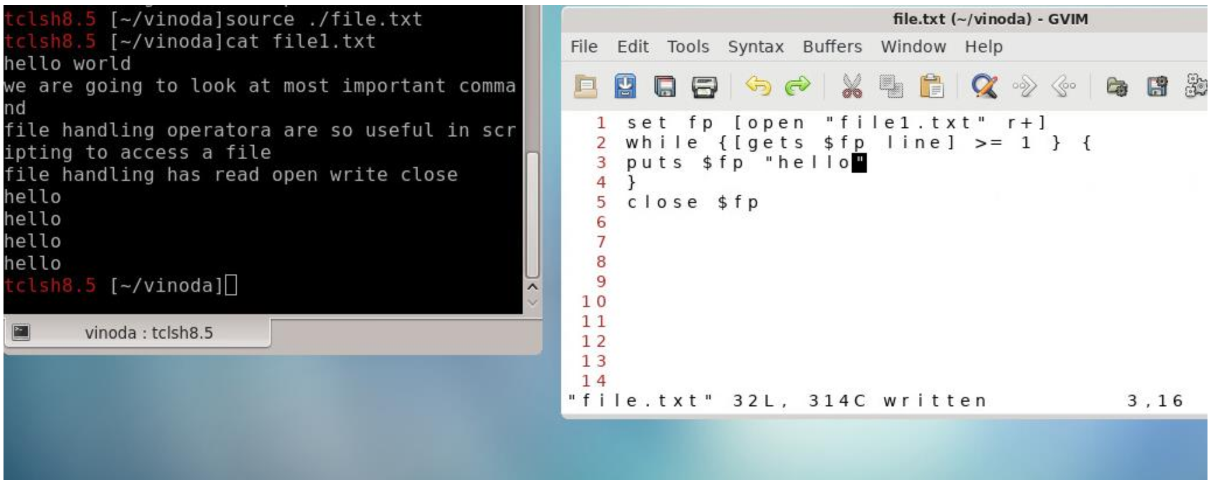
```
tclsh8.5 [~]proc test { } {
>upvar x y
>puts $y
>set y 4
>puts $y
>}
tclsh8.5 [~]set x 1
1
tclsh8.5 [~]puts $
$
tclsh8.5 [~]puts $x
1
tclsh8.5 [~]test
1
4
tclsh8.5 [~]puts $x
4
tclsh8.5 [~]
```

Global scope:
The variable is called outside the definition.
Syntax: global <variable>

```
Eg: proc test { } {
        Global x
        Puts “x is $x”
        Proc nested { } {
        Global x
        Puts “x is $x”
        }
        }
        Set x 1
        Test
        Nested
        Puts “x is $x”
```

```
Output: x is 1
           X is 1
           X is 1
```

File handling operations:
This is a text processing; each record is converted into a list. We can display or even edit a file.
Commands used are: open, read, close, puts, gets
Puts: to display the content
Puts <user defined channel id> <variable/content>
Gets: it converts records into list and prints the number of records.
Syntax: gets <user defined channel id> <variable>
Open: it opens the file
Syntax: open “<file name”> <access mode>
Close: it closes the file
Syntax: close $<used defined channel id>
Read: to read the content of file
Syntax: read <used defined channel id>
Access modes:
r: It helps in accessing the file in read mode.

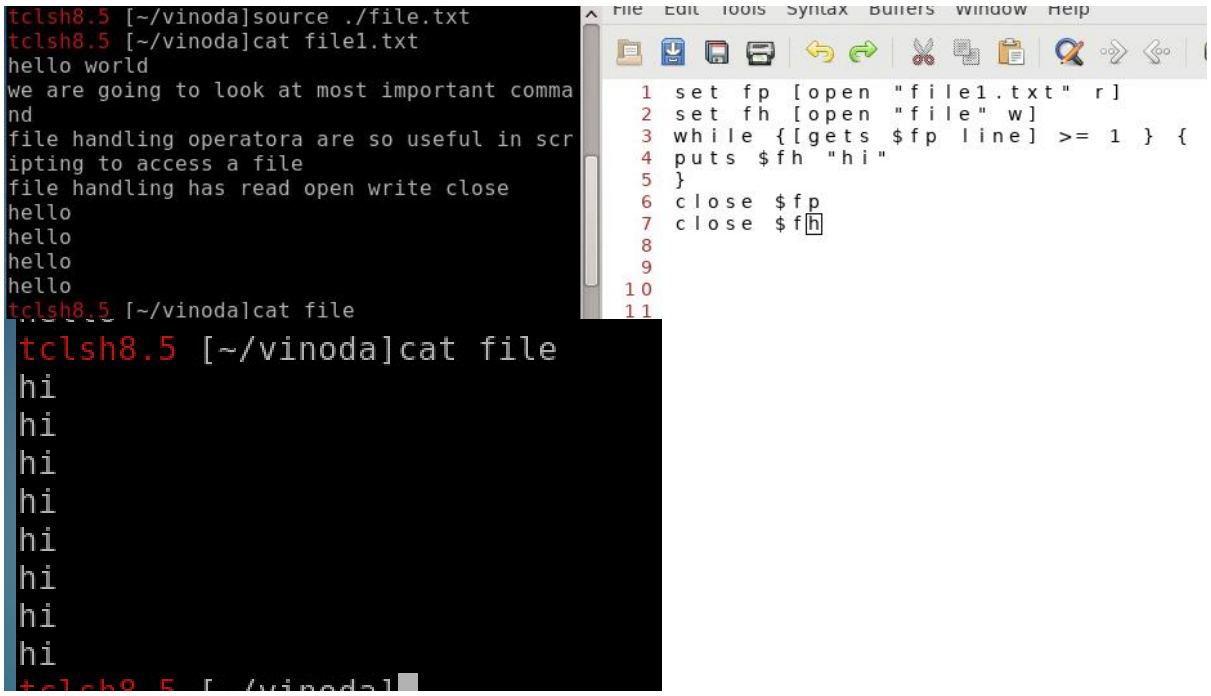


```
tclsh8.5 [~/vinoda]source ./file.txt
tclsh8.5 [~/vinoda]cat file1.txt
hello world
we are going to look at most important comma
nd
file handling operatora are so useful in scr
ipting to access a file
file handling has read open write close
hello
hello
hello
hello
tclsh8.5 [~/vinoda]
```

vinoda : tclsh8.5
file.txt (~/vinoda) - GVIM
File Edit Tools Syntax Buffers Window Help

```
set fp [open "file1.txt" r+]
while {[gets $fp line] >= 1 } {
puts $fp "hello"
}
close $fp
"file.txt" 32L, 314C written          3,16
```

w: It is used to write the content into the file, it creates a file if it does not exists, if exists it over writes the content.

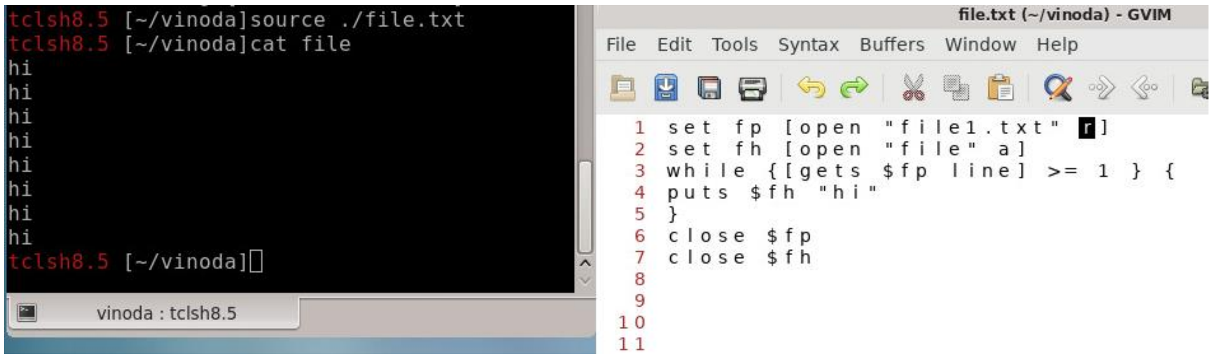


```
tclsh8.5 [~/vinoda]source ./file.txt
tclsh8.5 [~/vinoda]cat file1.txt
hello world
we are going to look at most important comma
nd
file handling operatora are so useful in scr
ipting to access a file
file handling has read open write close
hello
hello
hello
hello
tclsh8.5 [~/vinoda]cat file
```

```
set fp [open "file1.txt" r]
set fh [open "file" w]
while {[gets $fp line] >= 1 } {
puts $fh "hi"
}
close $fp
close $fh
```

```
tclsh8.5 [~/vinoda]cat file
hi
hi
hi
hi
hi
hi
hi
hi
```

a: It is used to append content to the file, the file should exist.

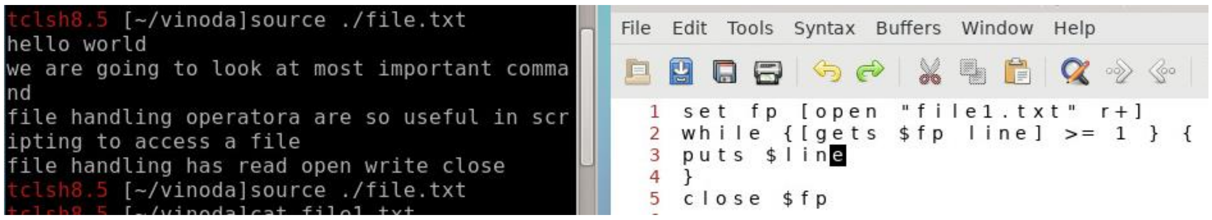


```
tclsh8.5 [~/vinoda]source ./file.txt
tclsh8.5 [~/vinoda]cat file
hi
hi
hi
hi
hi
hi
hi
hi
tclsh8.5 [~/vinoda]
```

```
set fp [open "file1.txt" r]
set fh [open "file" a]
while {[gets $fp line] >= 1 } {
puts $fh "hi"
}
close $fp
close $fh
```

vinoda : tclsh8.5
file.txt (~/vinoda) - GVIM
File Edit Tools Syntax Buffers Window Help

r+: It is used open the file in read mode and even helps in writing content in the file. The file should be existing.



```
tclsh8.5 [~/vinoda]source ./file.txt
hello world
we are going to look at most important comma
nd
file handling operatora are so useful in scr
ipting to access a file
file handling has read open write close
tclsh8.5 [~/vinoda]source ./file.txt
```

```
set fp [open "file1.txt" r+]
while {[gets $fp line] >= 1 } {
puts $line
}
close $fp
```

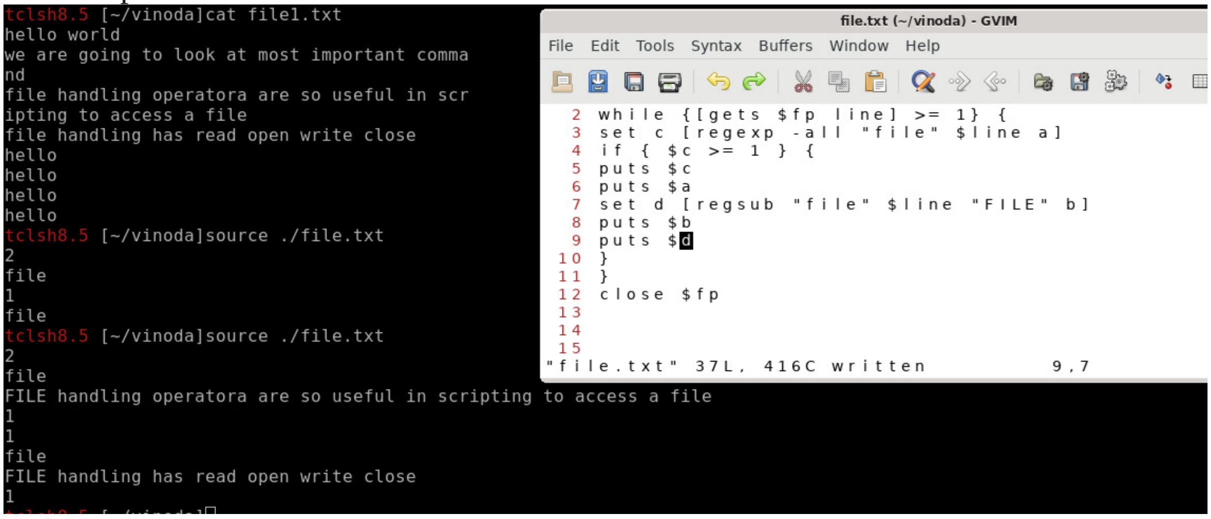
w+: It is used to write and read the file. If file does not exist it creates the file and overwrites the data.
a+: It is used to append the content on to the file and open in read mode. It creates the file if it does not exist.

Regular expressions:
It is useful when listing operations during search and replace.
regexp: It is used for matching patterns in a string.
regsub: It is used to replace, search and modify a file.
regexp:
syntax: regexp <options> “<pattern match>” “<string to match>” variable

output of regexp will be the numeric value of how many matches are found.
The most used options are : -inline, -all, -not.

regsub:
syntax: regsub <option> "<pattern match>" "<string to match>" "<string to replace>" <variable>
output of regsub is the numeric value of how many replacements have occurred.
Example on both regexp and regsub:

```
Set fh [open "file.txt" r]
Set fp [open "file1.txt" w]
While { [gets $fh line] >= 0} {
If { [regexp –all –inline "pass" $line  a] >= 1 } {
Puts $a
Puts [regsub –all "pass" $line "p" b]
Puts $b
} else {
Puts $fp $b
}
}
Close $fh
Close $fp
```

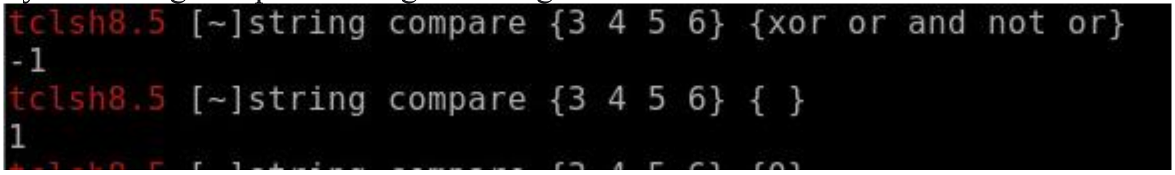


```
tclsh8.5 [~/vinoda]cat file1.txt
hello world
we are going to look at most important comma
nd
file handling operatora are so useful in scr
ipting to access a file
file handling has read open write close
hello
hello
hello
hello
tclsh8.5 [~/vinoda]source ./file.txt
2
file
1
file
tclsh8.5 [~/vinoda]source ./file.txt
2
file
FILE handling operatora are so useful in scripting to access a file
1
1
file
FILE handling has read open write close
1
```

file.txt (~/vinoda) - GVIM
File Edit Tools Syntax Buffers Window Help

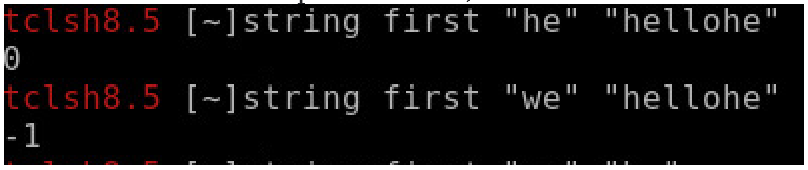
```
while {[gets $fp line] >= 1} {
set c [regexp -all "file" $line a]
if { $c >= 1 } {
puts $c
puts $a
set d [regsub "file" $line "FILE" b]
puts $b
puts $d
}
}
close $fp



"file.txt" 37L, 416C written          9,7
```

Strings: Strings are same as array or list command.
Creating a string: set {….}
String compare: If two string are same then print 0 if not prints -1 if string occurs 1st in dictionary order if not it prints 1.
Syntax: sring compare "string1" "string2"



```
tclsh8.5 [~]string compare {3 4 5 6} {xor or and not or}
-1
tclsh8.5 [~]string compare {3 4 5 6} { }
1
```

String first: It is used to search a word in string and access only the first occurrence.
Syantx: string first "word" "string"
If the word is found it prints 1 or 0, if not -1

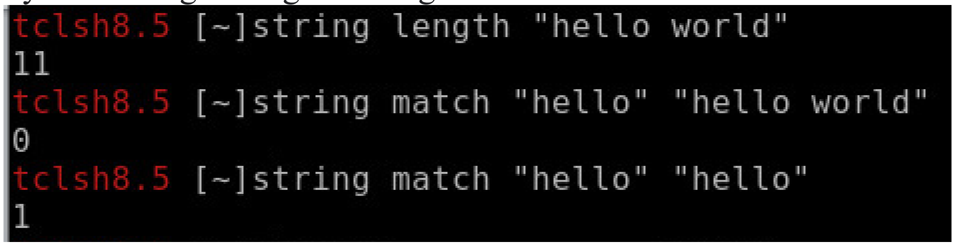


```
tclsh8.5 [~]string first "he" "hellohe"
0
tclsh8.5 [~]string first "we" "hellohe"
-1
```

Index: It prints me the number of indexes of the string, i.e. the number of characters of a string. It even includes space as one character.
Syntax: string index “string”

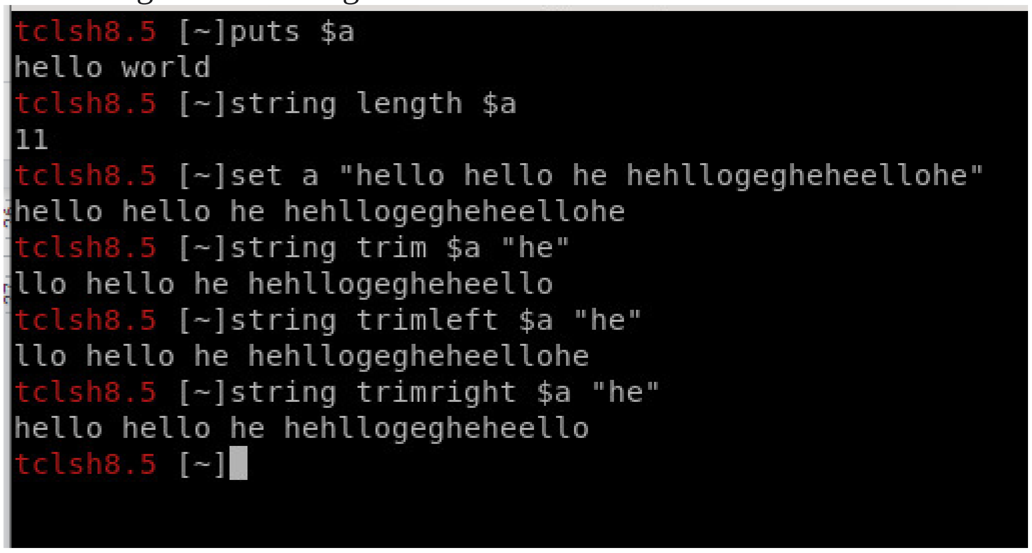
Length: The output of string length is similar to gets command. It prints the total length.
Syntax: string length “string”

Match pattern:
It matches the pattern with respect to the index. If the exact is not matched then it prints -1 if matches it prints 1.
Syntax: string “string1” “string2”



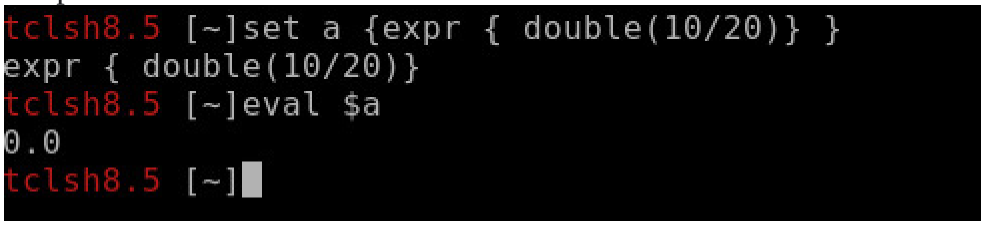
```
tclsh8.5 [~]string length "hello world"
11
tclsh8.5 [~]string match "hello" "hello world"
0
tclsh8.5 [~]string match "hello" "hello"
1
```

Trim:
Trims from both sides
Syntax: string trim “string” “trimming word”
To trim left side: trimleft
To trim right side: trimright



```
tclsh8.5 [~]puts $a
hello world
tclsh8.5 [~]string length $a
11
tclsh8.5 [~]set a "hello hello he hehllogegheheellohe"
hello hello he hehllogegheheellohe
tclsh8.5 [~]string trim $a "he"
llo hello he hehllogegheheello
tclsh8.5 [~]string trimleft $a "he"
llo hello he hehllogegheheellohe
tclsh8.5 [~]string trimright $a "he"
hello hello he hehllogegheheello
tclsh8.5 [~]
```

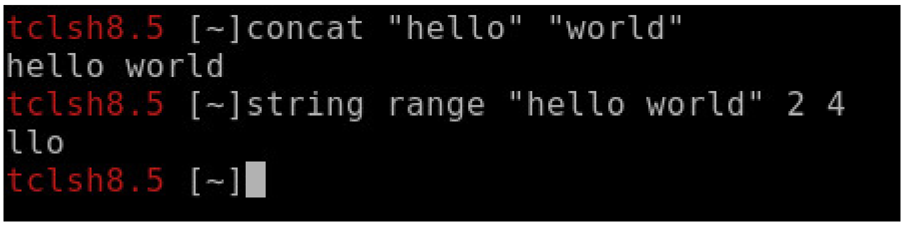
Eval: If command is present in a string and to be evaluated then use eval.
Eg: set a “puts hello”
Eval $a
Output: hello



```
tclsh8.5 [~]set a {expr { double(10/20)} }
expr { double(10/20)}
tclsh8.5 [~]eval $a
0.0
tclsh8.5 [~]
```

Concatenate: Two strings are joined using concat.
Syntax: concat “string1” “string2”

Range:
This returns the range of characters in string.
Syntax: string range <variable/string> <index start> <index end>



```
tclsh8.5 [~]concat "hello" "world"
hello world
tclsh8.5 [~]string range "hello world" 2 4
llo
tclsh8.5 [~]
```