Introduction

1.Introduction

1.1 History of UNIX

- UNIX is an Operating System (OS).
- UNIX was developed about 40 years ago i.e., 1969 at T& Bell Labs by Ken Thompson and Dennis Ritchie.
- It is a Command Line Interpreter.
- It was developed for the Mini-Computers as a time sharing system.
- UNIX was the predecessor of LINUX.

1.2 History of LINUX

- LINUX was created by Linus Torvalds in 1991.
- LINUX is a open source.
- LINUX is a variant of UNIX.

Introduction

1.3 Why LINUX/UNIX?

- LINUX is free.
- Can view and edit the source code of OS
- It is fully customizable.
- Most Important Feature is Stability
- Important in shared environments and critical applications
- LINUX has better security structure.
- High Portability
- Easy to port new H/W Platform
- Written in C which is highly portable

File Handling Utilities

File handling Utilities can be classified in to following categories They are ,

1) File creating and deleting
2) File naming or renaming
3) Editing files
4) File access permissions

1) File creating and deleting

Commands for creating and deleting the files are cat and rm respectively.

Creating: $ cat > file1

Enter your text

Ctrl+d

The file1 has been created

Commands

Deleting: rm COMMAND:

rm linux command is used to remove/delete the file from the directory.

SYNTAX: $rm [options..] [file | directory]

OPTIONS:

-f   Remove all files in a directory without prompting the user.

-i   Interactive. With this option, rm prompts for confirmation before removing any files.

-r (or) –R Recursively remove directories and subdirectories in the argument list. The directory will be emptied of files and removed. The user is normally prompted for removal of any write-protected files which the directory contains.

Ex:

```
        $ rm   file1
     The file1 has been deleted
```

Commands

EXAMPLE: 1) To Remove / Delete a file:

```
rm file1.txt
```

Here rm command will remove/delete the file file1.txt.

2. To delete a directory tree:

```
rm -ir tmp
```

This rm command recursively removes the contents of all subdirectories of the tmp directory, prompting you regarding the removal of each file, and then removes the tmp directory itself.

3. To remove more files at once

```
rm file1.txt file2.txt
```

rm command removes file1.txt and file2.txt files at the same time.

Commands

2) File naming or renaming: Unix support following command for renaming the file as,

```
$ mv  [options] file1 newfile
```

file1 renamed as newfile

mv command which is short for move. It is used to move/rename file from one directory to another. mv command is different from cp command as it completely removes the file from the source and moves to the directory specified, where cp command just copies the content from one file to another.

OPTIONS:

-f   This will not prompt before overwriting (equivalent to --reply=yes). mv -f will move the file(s) without prompting even if it is writing over an existing target.

-i   Prompts before overwriting another file.

Commands

EXAMPLE:

1. To Rename / Move a file:

```
mv file1.txt file2.txt
```

This command renames file1.txt as file2.txt

2. To move a directory

```
mv hscripts tmp
```

In the above line mv command moves all the files, directories and sub-directories from hscripts folder/directory to tmp directory if the tmp directory already exists. If there is no tmp directory it rename's the hscripts directory as tmp directory.

3. To Move multiple files/More files into another directory

```
mv file1.txt tmp/file2.txt newdir
```

This command moves the files file1.txt from the current directory and file2.txt from the tmp folder/directory to newdir.

File Handling Utilities

3) Editing files

Unix operating system allows the user to edit a file, with the help of vi editor. The user can append the data, update the existing data by using different commands of vi editor.

old file:$ vi f1.txt (edit the file)

```
   kljdf jfd
Kjgf jhkfg
```

Newfile:$ cat f1.txt (appending new line)

```
  kljdf jfd
Kjgf jhkfg
Hipo;iopte
```

File Handling Utilities

4) File access permissions

Unix operating system supports three types of users. They are

i) Owner ii)Group iii) others

Unix file system has three different file access permissions. They are,

i) Read ii) Write iii) Execute

http://sreenidhi.ed
/

The combination of file access permissions can be applied on above users by using **chmod** command

Syntax: $chmod a + rwx newfile

The new file can be accessed by any user. These file access permissions provides security for user files. In file maintaining system owner is the super user, so he can restrict other users from accessing the files.

Commands

ln COMMAND:

ln command is used to create link to a file (or) directory.

SYNTAX: ln [options] existingfile(or directory)name newfile(or directory)name

$ ln history for the file history in the form of the name indianhistory.

Types:

1 sybolic link

2. Hard Link

Symbolic link: it is coated by using –s option. It represents logical file some where else in the system.

Syntax:

```
$ ln –s file1 dir1
```

Commands

Hard Link : Hard links have same inode number

Syntax

```
$ ln  file1  file2
```

```
$ ln sample1 sample2
```

➢ Display inodes for both files using i argument of the ls command

```
$ ls -il sample1 sample2
```

```
1482256 -rw-r--r-- 2 bruno bruno 21 May 5 15:55 sample1
1482256 -rw-r--r-- 2 bruno bruno 21 May 5 15:55 sample
```

Commands

Unlink COMMAND:

unlink - delete a name and possibly the file it refers to

RETURN VALUE

On success, zero is returned. On error, -1 is returned, and *errno* is set appropriately.

Syntax

unlink FILE

Commands

mkdir COMMAND:

mkdir command is used to create one or more directories.

SYNTAX: mkdir [options] directories

OPTIONS:

-m	Set the access mode for the new directories.
-p	Create intervening parent directories if they don't exist.
-v	Print help message for each directory created.

EXAMPLE:

1. Create directory:

```
mkdir test
```

The above command is used to create the directory 'test'.

2. Create directory and set permissions:

```
 mkdir -m 666 test
```

The above command is used to create the directory 'test' and set the read and write permission.

3 .   -p : Create intervening parent directories if they don't exist.

```
Ex:  $ : mkdir -p student/marks/attendance/labmarks
```

4. v   Print help message for each directory created

```
Ex:  $ : mkdir  -v mamta

     mkdir: created directory 'mamata'
```

So it display a message that the directory been created.

Commands

rmdir COMMAND:

rmdir command is used to delete/remove a directory and its subdirectories.

SYNTAX: The Syntax is rmdir [options..] Directory OPTIONS: -p Allow users to remove the directory dirname and its parent directories which become empty.

EXAMPLE:

1. To delete/remove a directory

```
rmdir tmp
```

rmdir command will remove/delete the directory tmp if the directory is empty.

2. To delete a directory tree:

```
rm -ir tmp
```

This command recursively removes the contents of all subdirectories of the tmp directory, prompting you regarding the removal of each file, and then removes the tmp directory itself.

Commands

cp COMMAND:

cp command copy files from one location to another. If the destination is an existing file, then the file is overwritten; if the destination is an existing directory, the file is copied into the directory (the directory is not overwritten).

SYNTAX: $cp [OPTIONS]... SOURCE DEST

cp [OPTIONS]... SOURCE... DIRECTORY

cp [OPTIONS]... --target-directory=DIRECTORY SOURCE...

OPTIONS:

--a   same as -dpR.

--backup[=CONTROL]   make a backup of each existing destination file

-b like --backup but does not accept an argument.

-f   if an existing destination file cannot be opened, remove it and try

Commands

Ulimit COMMAND:

Ulimit stands for user limit.

Ulimit gives a value as an output, which specifies the largest file that can be created by the user in the file system.

Syntax:

```
$ ulimit
3045678
```

The output value represents that user can’t create a file which is bigger than 3045678 bytes.

Find COMMAND:

Find command recursively check’s the directory tree to find out the given files either by name or by matching one or more file attributes.

Syntax:

```
Find path list selection_criteria action
```

Commands

Find /oracle -name temp -print

/oracle -→ path list

-name temp-→ selection_criteria

-print → action

Example1: find all the files in /home with name test.txt. Here –name is used to specify the filename.

find /home –name test.txt

Example2: find the files whose name is test.txt and in present working directory

find . –name test.txt Or

find –name test.txt

Example3: find all the files whose name contains both capital letters and small letters in it.

find /home –iname test.txt

-iname option is used to mention ignore the case sensitivity of a file.

Commands

Example4:Search for a file name test.txt and its permissions are 775 in a given box

find / -perm 775 –name test.txt

Example5: How about searcing files with SUID bit set and file permissions are 755?

find / -perm 4755

Example6: Search for all the files with name test.txt and the owner of this file is amar

find / -user Surendra –name test.txt

Example7: find all the files whos name is test.txt and owned by a group called redcluster

find / -group redcluster –name test.txt

Commands

Pwd COMMAND:

Short for print working directory the pwd command displays the name of the current working directory.

Syntax

pwd

Examples

pwd

Typing pwd at the prompt would give you something similar to:

/home/computerhope/public_html

Security by file permissions

File ownership is an important component of UNIX that provides a secure method for storing files. Every file in UNIX has the following attributes:

Owner permissions: The owner's permissions determine what actions the owner of the file can perform on the file.

Group permissions: The group's permissions determine what actions a user, who is a member of the group that a file belongs to, can perform on the file.

Other (world) permissions: The permissions for others indicate what action all other users can perform on the file.

The Permission Indicators:

While using **ls -l** command it displays various information related to file permission as follows:

```
$ls -l
-rwxr-xr--  1 root   users 1024  Nov 2 00:10  myfile
drwxr-xr--- 1 root   users 1024  Nov 2 00:10  mydir
```

Here first column represents different access mode ie. permission associated

Security by file permissions

File Access Modes:

The permissions of a file are the first line of defense in the security of a Unix system. The basic building blocks of Unix permissions are the **read**, **write**, and **execute** permissions, which are described below:

Read: Grants the capability to read ie. view the contents of the file.

Write: Grants the capability to modify, or remove the content of the file.

Execute: User with execute permissions can run a file as a program

Changing Permissions:

To change file or directory permissions, you use the **chmod** (change mode) command. There are two ways to use chmod: symbolic mode and absolute mode.

Using chmod in Symbolic Mode:

The easiest way for a beginner to modify file or directory permissions is to use the symbolic mode. With symbolic permissions you can add, delete, or specify

the permission set you want by using the operators in the following table.

Security by file permissions

Chmod operator	**Description**
+	Adds the designated permission(s) to a file or directory.
-	Removes the designated permission(s) from a file or directory.
=	Sets the designated permission(s).

Here's an example using testfile. Running ls -1 on testfile shows that the file's permissions are as follows:

```
$  ls -l testfile
```

```
-rwxrwxr--  1 amrood   users 1024  Nov 2 00:10  testfile
```

Then each example chmod command from the preceding table is run on testfile, followed by ls -l so you can see the permission changes:

Security by file permissions

```
$chmod o+wx testfile
$ls -l testfile
-rwxrwxrwx  1 root   users 1024  Nov 2 00:10  testfile

$chmod u-x testfile
$ls -l testfile
-rw-rwxrwx  1 root   users 1024  Nov 2 00:10  testfile

$chmod g=r-x testfile
$ls -l testfile
-rw-r-xrwx  1 root   users 1024  Nov 2 00:10  testfile
```

Security by file permissions

Here's how you could combine these commands on a single line:

```
$chmod o+wx,u-x,g=r-x testfile
$ls -l testfile
-rw-r-xrwx  1 root   users 1024  Nov 2 00:10  testfile
```

Using chmod with Absolute Permissions:

The second way to modify permissions with the chmod command is to use a number to specify each set of permissions for the file.

Each permission is assigned a value, as the following table shows, and the total of each set of permissions provides a number for that set.

Security by file permissions

Number	Octal Permission Representation	Ref
0	No permission	---
1	Execute permission	--x
2	Write permission	-w-
3	Execute and write permission: 1 (execute) + 2 (write) = 3	-wx
4	Read permission	r--
5	Read and execute permission: 4 (read) + 1 (execute) = 5	r-x
6	Read and write permission: 4 (read) + 2 (write) = 6	rw-
7	All permissions: 4 (read) + 2 (write) + 1 (execute) = 7	rwx

Security by file permissions

Here's an example using testfile. Running ls -1 on testfile shows that the file's permissions are as follows:

```
$ls -l testfile
-rwxrwxr--  1 root   users 1024  Nov 2 00:10  testfile
```

Then each example chmod command from the preceding table is run on testfile, followed by ls -l so you can see the permission changes:

```
$ chmod 755 testfile
$ls -l testfile
-rwxr-xr-x  1 root   users 1024  Nov 2 00:10  testfile
$chmod 743 testfile
$ls -l testfile
-rwxr---wx  1 root   users 1024  Nov 2 00:10  testfile
$chmod 043 testfile
$ls -l testfile
----r---wx  1 root   users 1024  Nov 2 00:10  testfile
```

Security by file permissions

Changing Owners and Groups:
While creating an account on Unix, it assigns a owner ID and a group ID to each user. All the permissions mentioned above are also assigned based on Owner and Groups.
Two commands are available to change the owner and the group of files:
chown: The chown command stands for "change owner" and is used to change the owner of a file.
chgrp: The chgrp command stands for "change group" and is used to change the group of file.
Changing Ownership:
The chown command changes the ownership of a file. The basic syntax is as follows:

```
$ chown user filelist
```

The value of user can be either the name of a user on the system or the user id (uid) of a user on the system.
Following example:

```
$ chown root1 testfile
```

Changes the owner of the given file to the user **root1**
NOTE: The super user, root, has the unrestricted capability to change the ownership of a any file but normal users can change only the owner of files they own.

Security by file permissions

- **Changing Group Ownership:**
- The chrgp command changes the group ownership of a file. The basic syntax is as follows:
- $ chgrp group filelist   The value of group can be the name of a group on the system or the group ID (GID) of a group on the system.
- Following example:
- $ chgrp special testfile
- Changes the group of the given file to **special** group.

Process utilities

ps COMMAND:

ps command is used to report the process status. ps is the short name for Process Status.

SYNTAX: The Syntax is ps [options]

OPTIONS:

-e	List information about every process now running.
-f	Generates a full listing.
–u	option followed by user-id displays the processes owned by the user-id.
-l	Generate a long listing.

- To know the process ID of the process

```
$ pidof bashee
  24157
```

Process utilities

EXAMPLE:

1. ps

Output:

```
PID TTY TIME CMD
2540 pts/1 00:00:00 bash
2621 pts/1 00:00:00 ps
```

In the above example, typing ps alone would list the current running processes.

2. ps -f

Output:

```
UID          PID    PPID   C      STIME  TTY    TIME     CMD
abc          2540   2536   0      15:31  pts/1  00:00:00 bash
xyz          2639   2540   0      15:51  pts/1  00:00:00
ps -f
```

Displays full information about currently running processes.

Process utilities

who COMMAND:

who command can list the names of users currently logged in, their terminal, the time they have been logged in, and the name of the host from which they have logged in.

SYNTAX: who [options] [file]

OPTIONS:

-H	Print column headings above the output.
-u	provides a detailed list

EXAMPLE:

1. who -uH

Output:

```
NAME  LINE   TIME          IDLE   PID    COMMENT
hiox  ttyp3  Jul 10 11:08  .      4578
```

This sample output was produced at 11 a.m. The "." indicates activity within the last minute.

Process utilities

who am i

who am i command prints the user name.

w command:

w - show who is logged on and what they are doing

DESCRIPTION

w displays information about the users currently on the machine, and their processes. The header shows, in this order, the current time, how long the system has been running, how many users are currently logged on, and the system load averages for the past 1, 5, and 15 minutes.

Kill COMMAND:

• To terminate the running process

Kill command in UNIX and Linux is normally used to kill a suspended or hanged process or process group.

Example:

“if you issue the ps command and find that one of your processes is hung or if you

Process utilities

The following example shows how to kill a process,

```
$ps –ef | grep sales

Sales 19336  19334  0    05:24:32           pts/4  0:01  -ksh

Sales 19426  19336  0    06:01:01           pts/4  0:00  ora
```

Kill 19426   This will kill the process ora with PID 19426

To terminate more than one process at a time, use the following syntax

```
$kill signal PID PID PID

$pkill signal process process process
```

In kill utility we will give PID where as in pkill utility we have to give the name of the process

kill -9 is used to forcefully terminate a process in Unix. Here is syntax of kill command in UNIX.

```
ps -ef| grep process_identifier // will give you PID
kill -9 PID
```

Job Control:

- Lets simultaneously run a few commands that takes some time to complete.

```
$ sleep 150 &
$ sleep 250 &
$ sleep 300 &
$ sleep 200 &
```

Here we have executed 4 sleep commands and all are started in the background as denoted by &.

List the running jobs

```
$ jobs
```

And the output is ...

```
[1] running sleep 150
 [2] running sleep 250
 [3] - running sleep 300
 [4] + running sleep 200
```

In the output above, the number within [and] is the job number (job ID). The job number is unique for each job.

Bring a job to the foreground

```
$ fg %job-number
```

➢To bring job number 2 to the foreground, you run the following command.

```
$ fg %2
```

Suspend the job

To suspend a job, you first bring the job to the foreground and then press the keys Ctrl + z.

Alternately, you can also suspend a job running in the background as follows.

```
$ stop %job-number
```

Send a job to the background

```
$ bg %job-number
```

To resume the suspended job 2 in the background, you can run the following command.

```
$ bg %2
```

List only running jobs

```
$ jobs –r
```

List only suspended jobs

```
$ jobs –s
```

Terminate a job

To terminate a job, you first bring the running job to the foreground, and then press the keys Ctrl + c.

You can also terminate a job without bringing it in the foreground just by passing the job number to kill.

```
$ kill %job-number
```

For example, to kill the 3rd job, you can do as follows.

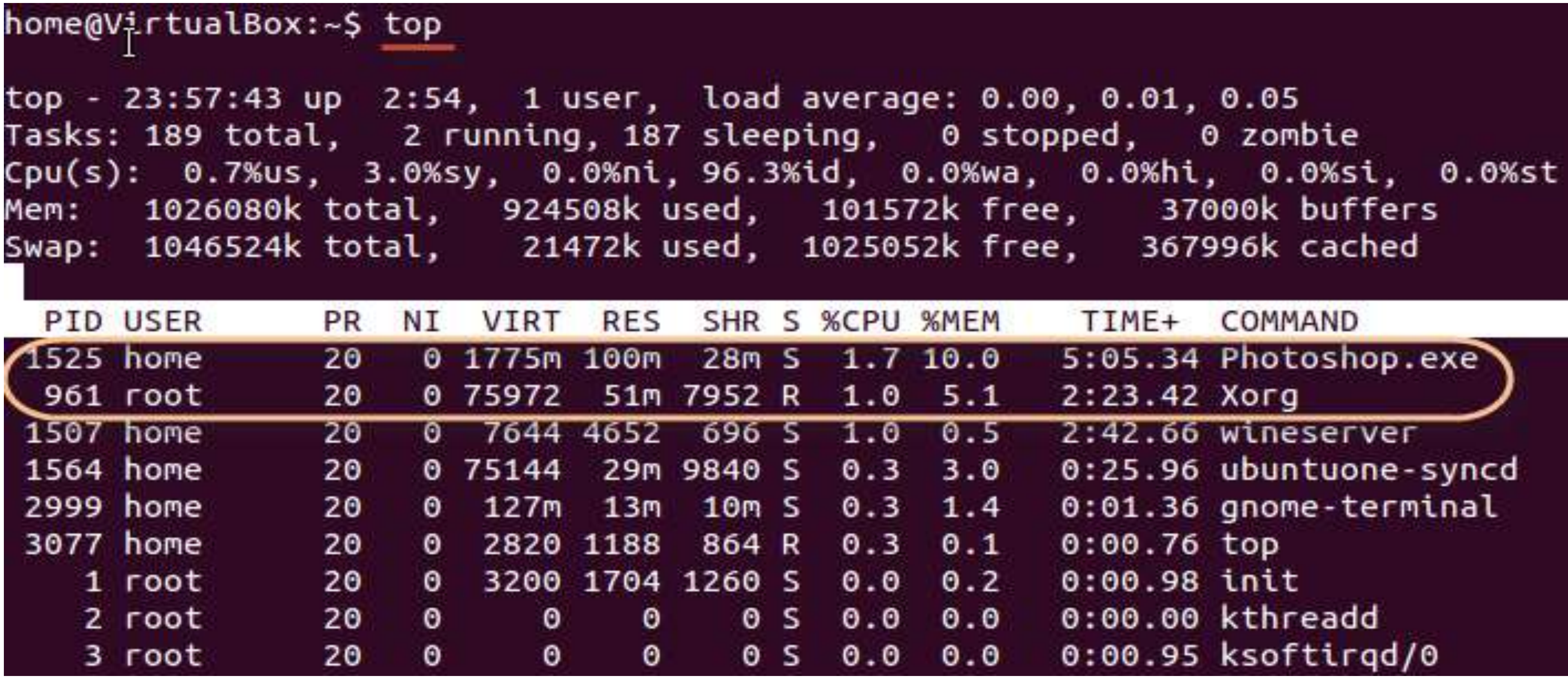
```
$ kill %3
```

Top command :

This utility tells the user about all the running processes on the Linux machine.

Display all the running processes

Syntax: $ top



```
home@VirtualBox:~$ top

top - 23:57:43 up  2:54,  1 user,  load average: 0.00, 0.01, 0.05
Tasks: 189 total,   2 running, 187 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.7%us,  3.0%sy,  0.0%ni, 96.3%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   1026080k total,   924508k used,   101572k free,    37000k buffers
Swap:  1046524k total,    21472k used,  1025052k free,   367996k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 1525 home      20   0 1775m 100m  28m S  1.7 10.0   5:05.34 Photoshop.exe
  961 root      20   0 75972  51m 7952 R  1.0  5.1   2:23.42 Xorg
 1507 home      20   0  7644 4652  696 S  1.0  0.5   2:42.66 wineserver
 1564 home      20   0 75144  29m 9840 S  0.3  3.0   0:25.96 ubuntuone-syncd
 2999 home      20   0  127m  13m  10m S  0.3  1.4   0:01.36 gnome-terminal
 3077 home      20   0  2820 1188  864 R  0.3  0.1   0:00.76 top
    1 root      20   0  3200 1704 1260 S  0.0  0.2   0:00.98 init
    2 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kthreadd
    3 root      20   0     0    0    0 S  0.0  0.0   0:00.95 ksoftirqd/0
```

Field	**Description**	**Example 1**	**Example 2**
PID	The process ID of each task	1525	961
User	The username of task owner	Home	Root
PR	Priority Can be 20(highest) or -20(lowest)	20	20
NI	The nice value of a task	0	0
VIRT	Virtual memory used (kb)	1775	75972
RES	Physical memory used (kb)	100	51
SHR	Shared memory used (kb)	28	7952
S	Status There are five types: 'D' = uninterruptible sleep 'R' = running 'S' = sleeping 'T' = traced or stopped 'Z' = zombie	S	R
%CPU	% of CPU time	1.7	1.0
%MEM	Physical memory used	10	5.1
TIME+	Total CPU time	5:05.34	2:23.42
Command	Command name	Photoshop.exe	Xorg

Nice:
•Starts a process with a given priority
▪Linux can run a lot of processes at a time, which can slow down the speed of some high priority processes and result in poor performance.
•To avoid this, you can tell your machine to prioritize processes as per your requirements.
•This priority is called Niceness in Linux, and it has a value between -20 to 19.
•The lower the Niceness index, the higher would be a priority given to that task.
•The default value of all the processes is 0.

To start a process with a niceness value other than the default value use the following syntax

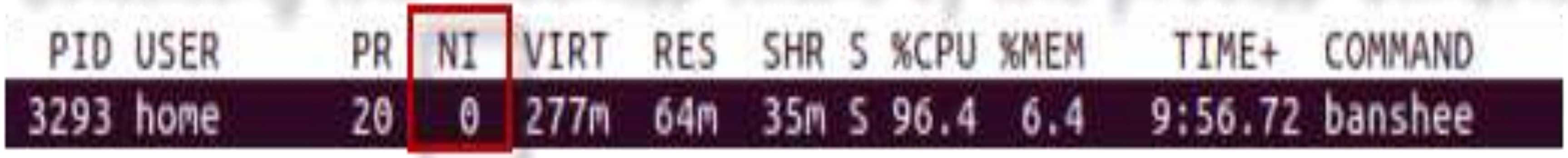
Syntax: $ nice -n 'Nice value' process name

•If there is some process already running on the system, then you can 'Renice' its value using syntax.

Syntax: renice 'nice value' -p 'PID'

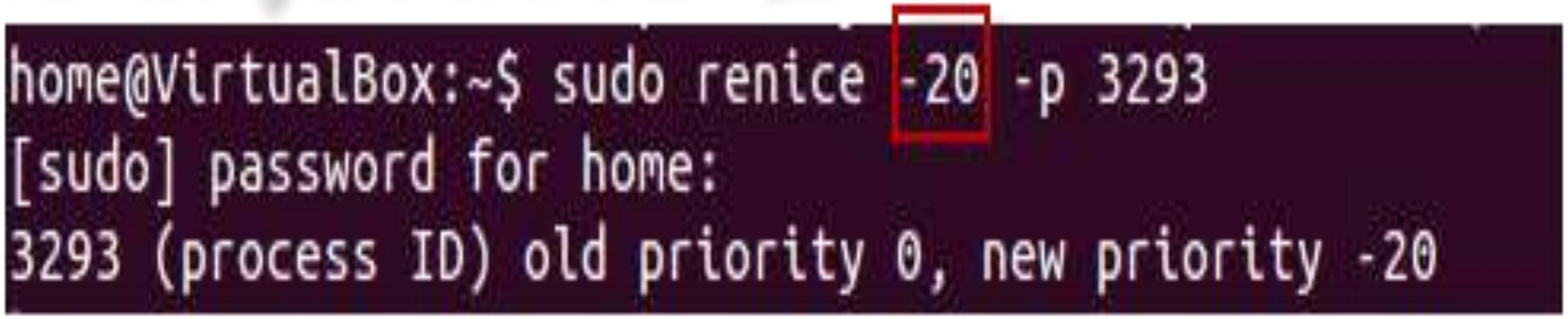
To change Niceness, you can use the 'top' command to determine the PID (process id) and its Nice value. Later use the renice command to change the value.

Checking the niceness value of the process 'banshee'



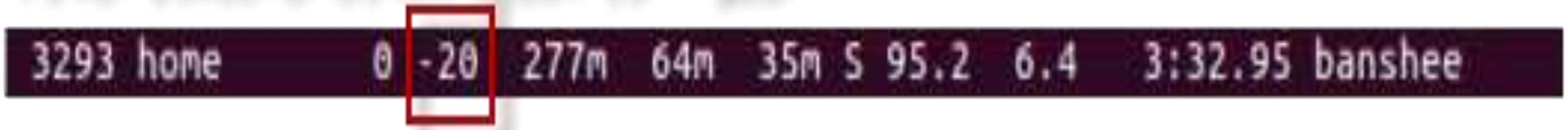
```
  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 3293 home      20   0  277m  64m  35m S 96.4  6.4   9:56.72 banshee
```

Renicing the value to -20



```
home@VirtualBox:~$ sudo renice -20 -p 3293
[sudo] password for home:
3293 (process ID) old priority 0, new priority -20
```

The value changed to -20



```
 3293 home       0 -20  277m  64m  35m S 95.2  6.4   3:32.95 banshee
```

nohup :

- When using the command shell, prefixing a command with **nohup** prevents the command from being aborted automatically when you log out or exit the shell.

- The name **nohup** stands for "no hangup." The hangup (**HUP**) signal, which is normally sent to a process to inform it that the user has logged off (or "hung up"), is intercepted by **nohup**, allowing the process to continue running.

-
```
$ nohup sleep 200&
21343

$ ps aux | grep sleep
```

Disk utilities

The utilities which will display the information about the disk is known as the disk utilities.

The following mentioned are the disk utilities

which are used in common,

1) df
2) du

df COMMAND:

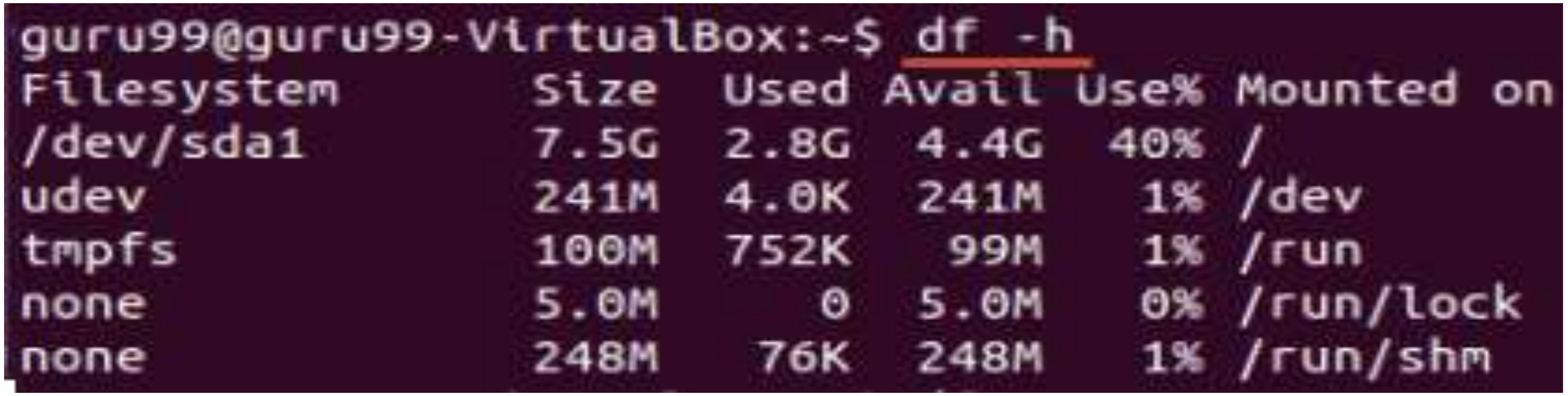
df command is used to find the empty disk space. Df stands for disk free.

Example:

```
 $ df
Filesystem          1K-blocks          Used       Available          Use%      Mounted on
/dev/sda1            132239776         6210884 119311504             5%          /
/tmpfs              4021876            0          4021876            0%          /dev/shm
/dev/sdb2           30969600           117740      29278696          1%        /home/oracle
/dev/sdc1           576310180          71232      546964104          1%          /home/data
```

• If you want the above information in a readable format, then use the command

$ df –h



```
guru99@guru99-VirtualBox:~$ df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda1       7.5G  2.8G  4.4G  40% /
udev            241M  4.0K  241M   1% /dev
tmpfs           100M  752K   99M   1% /run
none            5.0M     0  5.0M   0% /run/lock
none            248M   76K  248M   1% /run/shm
```

Free :

This command shows the free and used memory (RAM) on the Linux system.

• You can use the arguments

- free -m to display output in MB
- free -g to display output in GB

Disk utilities

Du COMMAND:

The du command displays the disk space used by the specified files and directories.

Syntax:

du [option]….[file]

Options:

1) –a: it writes counts for all files, not just for directories .
2) -b: it prints the size in bytes.
3) -c: it prints the grand total (i.e, the total disk space used by the directory).
4) -h: it prints the human readable formats.example1k,2g
5) -s: it does not show the size of sub-directories.

```
[mamta@localhost ~]$ du -a
8       ./.zshrc
8       ./f1
8       ./.bashrc
8       ./.kde/Autostart/.directory
16      ./.kde/Autostart
```

To see the file size

```
$ du –sh  new.tar

12k   new.tar
```

to compress the tar file use gzip

```
$ gzip –v new.tar
$ ls
  new.tar.gz
```

To check whether the file is compressed or not type

```
‘$ du –sh new.tar.gz

4.0k new.tar.gz
```

Networking commands

The commands which we will use for the communication between more than one system for sharing resources or any other issue are known as the networking commands.

Commands such as telnet, ftp, rlogin, finger are commonly used networking commands.

telnet COMMAND:

telnet is a internet protocol which enables user to connect a remote system and transfer data.

Syntax: $ telnet hostname or telnet ip_address

Example : *$ telnet itctnss0123*

```
Trying 164.130….
Connected to itctnss0123.
Escape character is '^]'.
SunOS 5.8
login: root
Password:
Last login: Thu Feb 24 16:13:12 from itctnss0123
Sun Microsystems Inc. SunOS 5.8 Generic Patch February 2004
```

Networking commands

Finger COMMAND:
The finger command is used to find out the details of a user, on both local and remote system.
Finger may be disabled on other systems for security reasons.
Following are the simple syntax to use finger command:
Check all the logged in users on local machine as follows:
Examples:

```
[amar@localhost ~]$ finger
Login     Name       Tty      Idle  Login Time   Office     Office Phone
amar                 pts/1          May 25 10:06 (172.16.10.5)
root      root       pts/0      7d  May 16 14:03 (:0.0)
```

```
[amar@localhost ~]$ finger amar
Login: amar                              Name: (null)
Directory: /home/amar                    Shell: /bin/bash
On since Sat May 25 10:06 (IST) on pts/1 from 172.16.10.5
No mail.
No Plan.
```

Networking commands

Ftp is the user interface to the **Internet** standard File Transfer Protocol. The program allows a user to transfer files to and from a remote network site.

This utility helps you to upload and download your file from one computer to another computer.

The ftp utility has its own set of UNIX like commands which allow you to perform tasks such as:

- Connect and login to a remote host.
- Navigate directories.
- List directory contents
- Put and get files
- Transfer files as ascii, ebcdic or binary

Syntax:

Following is the simple syntax to use **ping** command:

```
$ftp hostname or ip-address
```

Networking commands

```
[amar@localhost ~]$ ftp amar
ftp: amar: unknown host
ftp> quit
[amar@localhost ~]$ ftp 172.16.1.254
ftp: connect: Connection refused
ftp>
ftp> quit
```

Command	Description
put filename	Upload filename from local machine to remote machine.
get filename	Download filename from remote machine to local machine.
mput file list	Upload more than one files from local machine to remove machine.
mget file list	Download more than one files from remote machine to local machine.
prompt off	Turns prompt off, by default you would be prompted to upload or download movies using mput or mget commands.
prompt on	Turns prompt on.
dir	List all the files available in the current directory of remote machine.
cd dirname	Change directory to dirname on remote machine.
lcd dirname	Change directory to dirname on local machine.
quit	Logout from the current login.

Networking commands

rlogin COMMAND:

rlogin command helps the user to log on to his identical remote account, with out using either the username or password.

The "*rlogin*" command is similar to telnet. The rlogin starts a terminal session on a remote host.

Syntax : *$rlogin hostname*

Example : 1) [amar@localhost ~]$ rlogin 172.16.1.254

```
connect to address 172.16.1.254 port 543: Connection refused
Trying krb4 rlogin...
connect to address 172.16.1.254 port 543: Connection refused
trying normal rlogin (/usr/bin/rlogin)
```

2) [amar@localhost ~]$ rlogin amar

```
amar: host unknown
Trying krb4 rlogin...
amar: host unknown
trying normal rlogin (/usr/bin/rlogin)
amar: Unknown host
```

Filters

a filter is a program that accepts input, transforms it and outputs the transformed data. The idea of the filter is closely associated with several ideas that are part of the UNIX operating system: standard input and output, input/output redirection and pipes.

Standard input and output refer to default locations from which a program will take input and to which it will write output. The standard input (STDIN) for a program running interactively at the command line is the keyboard; the standard output (STDOUT) is the terminal screen.

With input/output redirection, a program can take input or send output using a location other than standard input or output—a file, for example. Redirection of STDIN is accomplished using the < symbol, redirection of STDOUT by >.

Filters

cat COMMAND:
cat linux command concatenates files and print it on the standard output.
SYNTAX: $ cat [OPTIONS] [FILE]...
OPTIONS:

-A	Show all.
-b	Omits line numbers for blank space in the output.
-e	A $ character will be printed at the end of each line prior to a new line.
-E	Displays a $ (dollar sign) at the end of each line.
-n	Line numbers for all the output lines.
-s	If the output has multiple empty lines it replaces it with one empty line.
-T	Displays the tab characters in the output.
-v	Non-printing characters (with the exception of tabs, new-lines and form-feeds) are printed visibly.

Examples:

1. To Create a new file:

```
cat > file1.txt
```

This command creates a new file file1.txt. After typing into the file press control+d (^d) simultaneously to end the file.

Filters

2. To Append data into the file:

```
cat >> file1.txt
```

To append data into the same file use append operator >> to write into the file, else the file will be overwritten (i.e., all of its contents will be erased).

3. To display a file:

```
cat file1.txt
```

This command displays the data in the file.

4. To concatenate several files and display:

```
cat file1.txt file2.txt
```

The above cat command will concatenate the two files (file1.txt and file2.txt) and it will display the output in the screen. Some times the output may not fit the monitor screen. In such situation you can print those files in a new file or display the file using less command.

```
cat file1.txt file2.txt | less
```

Filters

5. To concatenate several files and to transfer the output to another file.

```
cat file1.txt file2.txt > file3.txt
```

In the above example the output is redirected to new file file3.txt. The cat command will create new file file3.txt and store the concatenated output into file3.txt.

Display beginning and end of files

head COMMAND:

head command is used to display the first ten lines of a file, and also specifies how many lines to display

SYNTAX: $ head [options] filename

Filters

OPTIONS:

-n	To specify how many lines you want to display.
-n number sign	The number option-argument must be a decimal integer whose sign affects the location in the file, measured in lines.
-c number sign	The number option-argument must be a decimal integer whose sign affects the location in the file, measured in bytes.

EXAMPLE:

1. head f1.txt

   This command prints the first 10 lines of ' f1.txt '.

2. head -5 f1.txt

   The head command displays the first 5 lines of ' f1.txt '.

3. head -c 5 f1.txt

   The above command displays the first 5 characters of ' f1.txt '.

tail COMMAND:

Tail command is used to display the last or bottom part of the file. By default it displays last 10 lines of a file.

SYNTAX:

```
$ tail [options] filename
```

Filters

OPTIONS:

-b	To specify the units of blocks.
-n	To specify how many lines you want to display.
-c number	The number option-argument must be a decimal integer whose sign affects the location in the file, measured in bytes.
-n number	The number option-argument must be a decimal integer whose sign affects the location in the file, measured in lines.

EXAMPLE:

1. tail index.php
   It displays the last 10 lines of 'index.php'.
2. tail -2 index.php
   It displays the last 2 lines of 'index.php'.
3. tail -n 5 index.php
   It displays the last 5 lines of 'index.php'.
4. tail -c 5 index.php
   It displays the last 5 characters of 'index.php'.

Tail command also comes with an ‘+’ option which is not present in the head command. With this option tail command prints the data starting from specified line number of the file instead of end. For command: tail +n file_name, data will start printing from line number ‘n’ till the end of the file specified.

```
/mam123 $ cat hii
hii
how r u


fine thank you
bye

snist@snist-HP-Compaq-4000-Pro-SFF-PC:~/mam123$ tail +4 hii

fine thank you
bye
```

Filters

cut COMMAND:

cut command is used to cut out selected fields of each line of a file. The cut command uses delimiters to determine where to split fields.

SYNTAX:

cut [options]

OPTIONS:

-c	**Specifies character positions.**
-b	Specifies byte positions.
-d	flags Specifies the delimiters and fields.

EXAMPLE:

1. cut -c1-3 text.txt

Output:

```
Thi
```

Cut the first three letters from the above line.

2. cut -d, -f1,2 text.txt

Output:

```
This is, an example program
```

The above command is used to split the fields using delimiter and cut the first two fields.

```
mam123$ cat hii
hii
how r u


fine thank you
bye


snist@snist-HP-Compaq-4000-Pro-SFF-PC:~/mam123$ cut -c1 hii
h
h


f
b
snist@snist-HP-Compaq-4000-Pro-SFF-PC:~/mam123$ cut -c2 hii
i
o


i
y
```

```
snist@snist-HP-Compaq-4000-Pro-SFF-PC:~/mam123$ cut -c -2 hii
hi
ho


fi
by


snist@snist-HP-Compaq-4000-Pro-SFF-PC:~/mam123$ cut -c 2- hii
ii
ow r u


ine thank you
ye
```

```
  $ cat state.txt
Andhra Pradesh
Arunachal Pradesh
 Assam
Bihar
Chhattisgarh
```

```
$ cut -b 1-3,5-7 state.txt
Andra
Aruach
Assm
 Bihr
Chhtti
```

In this, 1- indicate from 1st byte to end byte of a line

```
            $ cut -b 1- state.txt
Andhra Pradesh
Arunachal Pradesh
 Assam
 Bihar
hhattisgarh
```

- In this, -3 indicate from 1st byte to 3rd byte of a line

```
 $ cut -b -3 state.txt
And
Aru
Ass
Bih
 Chh
```

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

Like in the file **state.txt** fields are separated by space if -d option is not used then it prints whole line:

```
$ cut -f 1 state.txt
Andhra Pradesh
 Arunachal Pradesh
Assam
Bihar C
hhattisgarh
```

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

```
$ cut -d " " -f 1 state.txt
```

```
Andhra
Arunachal
Assam
Bihar
 Chhattisgarh
```

❑ Command prints field from first to fourth of each line from the file. **Command:**

```
$ cut -d " " -f 1-4 state.txt
```

```
Andhra Pradesh
Arunachal Pradesh
Assam
Bihar
Chhattisgarh
```

Filters

paste COMMAND:

paste command is used to paste the content from one file to another file. It is also used to set column format for each line.

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

SYNTAX:

```
$ paste [options] [FILES]...
```

OPTIONS:

-s	Paste one file at a time instead of in parallel.
-d	Reuse characters from LIST instead of TABs .

EXAMPLE:

1. `paste test.txt>test1.txt`
   Paste the content from 'test.txt' file to 'test1.txt' file.
2. `ls | paste - - - -`
   List all files and directories in four columns for each line.

```
$ cat numbers
1
2
3
4
5
$ cat state
Arunachal Pradesh
Assam
Andhra Pradesh
 Bihar
Chhattisgrah

$ cat capital
Tanagar
 Dispur
 Hyderabad
 Patna
Raipur

$ paste number state capital
1 Arunachal Pradesh       Itanagar
2 Assam                    Dispur
3 Andhra Pradesh           Hyderabad
 4 Bihar                    Patna
5 Chhattisgrah              Raipur
```

- **d (delimiter):** Paste command uses the tab delimiter by default for merging the files. The delimiter can be changed to any other character by using the **-d** option. If more than one character is specified as delimiter then paste uses it in a circular fashion for each file line separation.

Only one character is specified

```
$ paste -d "|" number state capital
```

```
1|Arunachal Pradesh|Itanagar
2|Assam|Dispur
3|Andhra Pradesh|Hyderabad
4|Bihar|Patna
 5|Chhattisgrah|Raipur
```

More than one character is specified

```
$ paste -d "|," number state capital
1|Arunachal Pradesh,Itanagar
2|Assam,Dispur
3|Andhra Pradesh,Hyderabad
 4|Bihar,Patna
5|Chhattisgrah,Raipur
```

❑ **-s (serial):** We can merge the files in sequentially manner using the -s option. It reads all the lines from a single file and merges all these lines into a single line with each line separated by tab. And these single lines are separated by newline.

```
$ paste -s number state capital
1 2 3 4 5
Arunachal Pradesh Assam Andhra Pradesh Bihar Chhattisgrah
Itanagar Dispur Hyderabad Patna Raipur
```

✓ In the above command, first it reads data from **number** file and merge them into single line with each line separated by tab. After that newline character is introduced and reading from next file i.e. **state** starts and process repeats again till all files are read.

❑**Combination of -d and -s:**

The following example shows how to specify a delimiter for sequential merging of files:

```
$ paste -s -d ":" number state capital
```

```
1:2:3:4:5
Arunachal Pradesh:Assam:Andhra Pradesh:Bihar:Chhattisgrah
Itanagar:Dispur:Hyderabad:Patna:Raipur
```

sort COMMAND:

- sort command is used to sort the lines in a text file.
- SORT command sorts the contents of a text file, line by line.
- The sort command is a command line utility for sorting lines of text files. It supports sorting alphabetically, in reverse order, by number, by month and can also remove duplicates.

SYNTAX:

Sort [options] filename

OPTIONS:

-r	Sorts in reverse order.
-u	If line is duplicated display only once.
-o filename	Sends sorted output to a file.

```
$ cat > file.txt
abhishek
chitransh
satish
Rajan
 naveen
 divyam
 harsh

$ sort file.txt

Output :
abhishek
chitransh
divyam
harsh
naveen
rajan
satish
```

Sort function with mix file i.e. uppercase and lower case : When we have a mix file with both uppercase and lowercase letters then first the lower case letters would be sorted following with the upper case letters .
Example:
Create a file mix.txt

```
$ cat > mix.txt
abc
apple
BALL
Abc
Bat

$ sort mix.txt
abc
Abc
 apple
Bat
 BALL
```

-o Option : Unix also provides us with special facilities like if you want to write the **output to a new file**, output.txt, redirects the output like this or you can also use the built-in sort option -o, which allows you to specify an output file.
Using the -o option is functionally the same as redirecting the output to a file.

Syntax :

```
$ sort inputfile.txt > filename.txt
$ sort -o filename.txt inputfile.txt
```

Ex:

```
$ sort file.txt > output.txt
$ sort -o output.txt file.txt
$ cat output.txt
abhishek
chitransh
Divyam
harsh
naveen
rajan
satish
```

Filters

tr COMMAND:

- tr filter translates specified characters into other characters.

Syntax:

```
tr [options] string1 string2
```

Options

-c : complements the set of characters in string.i.e., operations apply to characters not in the given set

-d : delete characters in the first set from the output.

-s : replaces repeated characters listed in the set1 with single occurrence

- Translates any character read from stdin that is in string1 into the corresponding character from string2; otherwise the character is simply passed through.
- string1 and string2 normally have the same number of characters.
- Eg tr abc X5Z

  translates a into X, b into 5 and c into Z.

```
 $ cat  > f1
a
B
C
D
E
F
G
H
i
```

$ cat f1 | tr ‘a-i’ ‘1-9’

o/p:

$ echo “ welcome to linux” | tr ‘w’ ‘x’

```
o/p :
   xelcome to linux
```

❑**How to convert lower case to upper case**

```
$ cat > sort123
WELCOME TO
     linux programming

$ cat  sort123  | tr “[a-z]” “[A-Z]”
WELCOME TO
    LINUX PROGRAMMING
```

(OR)

```
$cat SORT123 | tr “[:lower:]” “[:upper:]”

$ cat SORT123 | tr “[:lower:]” “[:upper:]” “[:upper:]” “[:lower:]”
```

❑**How to translate white-space to tabs**

```
$ echo "Welcome To  LIINUX " | tr [:space:] '\t'
```

How to delete specified characters using -d option
To delete specific characters use the -d option.This option deletes characters in the first set specified.

```
$ echo "Welcome To  LINUX" | tr -d 'w‘
```

Output:

```
elcome To LINUX
```

To remove all the digits from the string, use

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

Output:

```
my ID is
```

❑ How to complement the sets using -c option

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

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

Output:

```
73535
```

-s (option) : : Replaces repeated characters listed in the set1 with single occurrence

```
$ cat > f2
 welcome to linux     programming
Linux     programming is interesting
Linux is an     operation syster
```

```
$ cat f2 |  tr –s [+m]
```

o/p:

❑ **How to squeeze multiple blank spaces to a single blank space**

```
$ cat f2 | tr –s ‘[:blank:] ‘
```
o/p:

Filters

uniq COMMAND:

- uniq assumes lines of ASCII file are sorted and deals with unique lines depending on the option given.
- No option, one copy of each unique line is printed to stdout. Duplicated lines are discarded.

Eg $ uniq dups

```
dups:    default output:
A         a
b        b
b        c
c        dd
dd       ddd
ddd
ddd
Ddd
```

- -u option prints only those lines that have no duplicates.

Eg uniq -u dups

```
dups:     -u output:
a          a
b          c
b         dd
```

Filters

-d option prints only those lines that have duplicates.

Eg uniq -d dups

```
output:
a        b
b        ddd
b
c
dd
ddd
ddd
ddd
```

uniq runs as filter or as program driving a pipeline.

There are options to compare fields and to count duplicated lines.

Filters

Wc COMMAND:

The purpose of wc command is to count the number of lines,words,characters in a particular file or files

Syntax: $ wc filename

Example:

```
$ wc file1
5   40    200       file1
```

First column represents number of lines

Second column represents number of words

Third column represents number of charcters

Fourth column shows the file name

OPTIONS:

-l (line)

-w (words)

-c (character)

Filters

cmp COMMAND:

Compares two files and tells you what line numbers are different.

By default location of first mismatch is displayed

Syntax: $ cmp [-options] filename1 filename2

-l	this option displays the list of all differences present in the file byte-by-byte.
-s	Write nothing for differing files; return exit status('0' or '1') only. the '0' specifies that two files are similar and '1' specifies that there is at least one that is different.

Text processing utilities

- The operations performed by the text processing utilities are as follows
- Processing the text
- Modifying the content of files
- Deleting words and charlacters form files

cat: cat is used to create the files.

$cat> filename

Type some text here

Press ctrl+d

$

Cat can also be used to display the contents Of a file.

$cat filename

Cat can also concatenate the contents of 2 files and store them in third file.

Cat>file1 file2>new file

Text processing utilities

tail:

tail command displays the end of the file. It displays the last ten lines by default.

```
$tail file
```

To display last 3 lines use

```
$tail –n 3 file        or
$tail -3 file
```

We can also address the lines from the beginning of the file instead of the end. The + count allows to do that.

Text processing utilities

head:

head command as the name implies, displays the top of the file. When used without an option, it displays the first 10 lines of the file.

```
 $head file
```

We can use –n option to specify a line count and display, say first 3 lines of the file.

```
$head –n 3 file  or $head -3 file
```

Sort:

Sort can be used for sorting the contents of a file.

```
   $sort shortlist
```

Sorting starts with the first character of each line and proceeds to the next character only when the characters in two lines are identical.

Sort options:

With –t option sorts a file based on the fields.

Text processing utilities

$sort –t “|” +2 shortlist

The sort order can be reversed with –r option.

Sorting on secondary key:

U can sort on more than one field i.e. u can provide a secondary key to sort.

If the primary key is the third field and the secondary key the second field, we can use

$sort –t \| +2 -3 +1 shortlist

Numeric sort (-n):

To sort on number field use sort with –n option.

$sort –t: +2 -3 –n group1

Removing duplicate lines (-u):

The –u option u purge duplicate lines from a file.

Text processing utilities

nl:

nl is used for numbering lines of a file.

Nl numbers only logical lines –those containing something other apart from the new line character.

```
$nl file
```

nl uses a tab as a default delimiter, but we can change it with –s option.

```
$nl –s: file
```

nl won’t number a line if it contains nothing.

Grep: globally search for a regular expression and print.

Grep scans a file for the occurrence of a pattern and depending on the options used, displays

- Lines containing the selected pattern.
- Lines not containing the selected pattern (-v).
- Line numbers where pattern occurs (-n)
- No. of lines containing the pattern (-c)
- File names where pattern occurs (-l)

Syntax:

grep option pattern filename(s)

```
$cat > geekfile.txt
unix is great os. unix is opensource.
 unix is free os. learn operating system.
Unix linux which one you choose.
uNix is easy to learn.unix is a multiuser os
Learn unix .unix is a powerful.
```

1. **-i : Case insensitive search :** The -i option enables to search for a string case insensitively in the give file.

```
$grep -i "UNix" geekfile.txt
```

o/p

-c : Displaying the count of number of matches : We can find the number of lines that matches the given string/pattern

```
$grep -c "unix" geekfile.txt
```

Output:

-l : Display the file names that matches the pattern : We can just display the files that contains the given string/pattern.

```
$grep -l "unix"  *
```

Or

```
 $grep -l "unix" f1.txt  f2.txt  f3.xt  f4.txt
```

Output:

```
geekfile.txt
```

-o : Displaying only the matched pattern : By default, grep displays the entire line which has the matched string. We can make the grep to display only the matched string by using the -o option.

```
$ grep -o "unix" geekfile.txt
```

Output:

```
unix
unix
unix
unix
```

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

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

Output:

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

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

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

Output:

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

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

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

Output:

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

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

```
$ grep "os$" geekfile.txt
```

Text processing utilities

Egrep: extended grep

Egrep extended set includes 2 special characters + and ?.

+ --matches one or more occurrences of the pervious character.

?-- matches zero or more occurrences of the pervious character.

Create a table like below :

```
100 Thomas   Manager    Sales        $5,000
200 Jason    Developer  Technology   $5,500
300 Sanjay   Sysadmin   Technology   $7,000
400 Nisha    Manager    Marketing    $9,500
500 Randy    DBA        Technology   $6,000
```

- Search for Specific Characters :

The following example searches for either J, or N, or R.

```
$ egrep  [JNR] employee.txt
```

```
 200 Jason Developer  Technology    $5,500
400  Nisha Manager     Marketing       $9,500
 500 Randy DBA           Technology     $6,000
```

- Search for a Range

The following example searches the range 6-9. i.e It searches for 6, or 7, or 8, or 9

```
$ egrep [6-9] employee.txt
```

```
300 Sanjay  Sysadmin Technology    $7,000
400 Nisha    Manager   Marketing       $9,500
500 Randy   DBA            Technology   $6,000
```

- egrep OR Example

Pipe symbol is used for egrep OR. The following searches for either Marketing or DBA.

```
$ egrep 'Marketing|DBA' employee.txt
```

```
400 Nisha   Manager   Marketing    $9,500
500 Randy   DBA        Technology   $6,000
```

fgrep: fast grep

- If search criteria requires only sequence expressions, fgrep is the best utility.
- Fgrep supports only string patterns, no regular expressions.
- The **fgrep** command differs from the **grep** and **egrep** commands because it searches for a string instead of searching for a pattern that matches an expression.
- The **fgrep** command uses a fast and compact algorithm
- The $, *, [, |, (,), and \ characters are interpreted literally by the **fgrep** command, These characters are not interpreted as parts of a regular expression, as they are interpreted in the **grep** and **egrep** command.

✓ The entire string must be enclosed in single quotation mark ('...'). If no files are specified, the **fgrep** command assumes standard input.

Syntax : fgrep options string filename

(or)

grep –f pattern filename

- To extract all the lines that contain an apostrophe use fgrep as follows:

```
$fgrep “’” file
```

To search several files for a simple string of characters:

```
$ fgrep  "strcpy“  *.c
```

this searches for the string strcpy in all files in the current directory with names ending in the .c character string.

To display the names of files that contain a pattern:

```
$  fgrep  -l   "strcpy“  *.c
```

This searches the files in the current directory that end with .cand displays the names of those files that contain the strcpy string.

Text processing utilities

- **Cut:** slitting the file vertically

U can slice a file vertically with cut command.

- Cutting columns(-c):

Cut with –c option cuts the columns.

To extract first 4 columns of the group file :

```
$cut –c 1-4 group1
```

The specification –c 1-4 cuts columns 1 to 4.

Cutting fields:

To cut 1st and 3rd fields use

```
$cut –d: -f1,3 group1
```

Text processing utilities

- **Paste:** pasting files
- What u cut with the cut can be pasted back with paste command-but vertically rather than horizontally. u can view two files side by side by pasting them.
- To join two files calc.lst and result.lst use
- $paste –d= calc.lst result.lst
- **Join:**
- is a command in Unix-like operating systems that merges the lines of two sorted text files based on the presence of a common field.
- The join command takes as input two text files and a number of options. If no command-line argument is given, this command looks for a pair of lines from the two files having the same first field (a sequence of characters that are different from space), and outputs a line composed of the first field followed by the rest of the two lines.

➢**join** command is used to join the two files based on a **key field present in both the files**. The input file can be separated by white space or any delimiter.

Syntax: **join [OPTION] FILE1 FILE2**

Ex:

```
$cat file1.txt
1 AAYUSH
2 APAAR
3 HEMANT
 4 KARTIK

 // displaying contents of second file //
$cat file2.txt
1 101
2 102
 3 103
 4 104
```

Now, in order to combine two files the files must have some common field. In this case, we have the numbering 1, 2... as the common field in both the files.
NOTE : When using join command, both the input files should be sorted on the KEY on which we are going to join the files.

```
$   join file1.txt file2.txt
1 AAYUSH   101
2 APAAR      102
 3 HEMANT   103
 4 KARTIK     104
```

// by default join command takes the first column as the key to join as in the above case //

using -a FILENUM option : Now, sometimes it is possible that one of the files contain extra fields so what join command does in that case is that by default, it only prints pairable lines. For example, even if file file1.txt contains an extra field provided that the contents of file2.txt are same then the output produced by join command would be same:

```
$cat file1.txt
1 AAYUSH
2 APAAR
3 HEMANT
 4 KARTIK
 5 DEEPAK
//displaying contents of file2.txt//
$cat file2.txt
1 101
2 102
3 103
4 104
```

$join file1.txt file2.txt

```
1 AAYUSH 101
2 APAAR    102
 HEMANT   103
4 KARTIK    104
```

// although file1.txt has extra field the output is not affected cause the 5 column in file1.txt was unpairable with any in file2.txt//

/using join with -a option// //1 is used with -a to display the contents of first file passed//

$join file1.txt file2.txt -a 1

```
 1 AAYUSH 101
2 APAAR 102
3 HEMANT 103
4 KARTIK 104
 5 DEEPAK
```

using -v option : Now, in case you only want to print unpairable lines *i.e* suppress the paired lines in output then **-v option** is used with join command.

```
$join file1.txt file2.txt -v 1
```

```
5 DEEPAK
```

using -1, -2 and -j option : As we already know that join combines lines of files on a common field, which is first field by default.However, it is not necessary that the common key in the both files always be the first column.join command provides options if the common key is other than the first column.

•The -1 and -2 here represents he first and second file and these options requires a numeric argument that refers to the joining field for the corresponding file.

•

```
$cat file1.txt
AAYUSH   1
APAAR     2
HEMANT   3
KARTIK    4

cat file2.txt
101   1
102   2
103   3
104   4

$join -1 2 -2 2 file1.txt file2.txt
1 AAYUSH  101
2 APAAR     102
3 HEMANT  103
4 KARTIK    104
```

//here -1 2 refers to the use of 2 column of first file as the common field and -2 2 refers to the use of 2 column of second file as the common field

Back up utilities

The two back up utilities are used in Linux. They are,

1) Cpio
2) Tar

cpio COMMAND: copy input-output

- Cpio command copies files to and from a backup device. It uses standard input to take the list of filenames.
- It then copies them with their contents and headers into stream which can be redirected to a file or a device.
- Cpio can be used with redirection and piping.
- Cpio uses two options-o (output) and –i (input) either of which must be there in the command line.

Back up utilities

tar COMMAND : the tape archive program

- Tar doesn’t normally write to the standard output but creates an archive in the media.
- Tar accepts file and directory names as arguments.
- It operates recursively.
- It can create several versions of same file in a single archive.
- It can append to an archive without overwriting the entire archive.
- -c option is used to copy files to backup device.
- $tar –cvf /dev/rdsk/foq18dt /home/sales/sql/*.sql
- The verbose option (-v) shows the no. of blocks used by each file.
- Files are restored with the –x (extract) key option. when no file or directory name is specified it restores all files from the backup device.

To create a tar file

```
$ mkdir dir12
$ cd dir12
dir12$  touch f1 f2 f3 f4 f5 f6

$ tar –cvf  dir123.tar dir12

$ ls
Dir123.tar
```

To see the file size

```
$ du –sh dir123.tar
120k  dir123
```

To compress tar file gzip

```
$ gzip dir123.tar
```

To compress tar.gz file use bzip2

```
$ bzip2 –v dir123.tar.gz
$ dir123.tar.gz.bz2
```

To extract bz2 files

```
$bunzip –v dir123.tar.gz.bz2
```

To extract gz files

```
$gunzip –v dir123.tar.gz
Dir123.tar
```

To view tar file

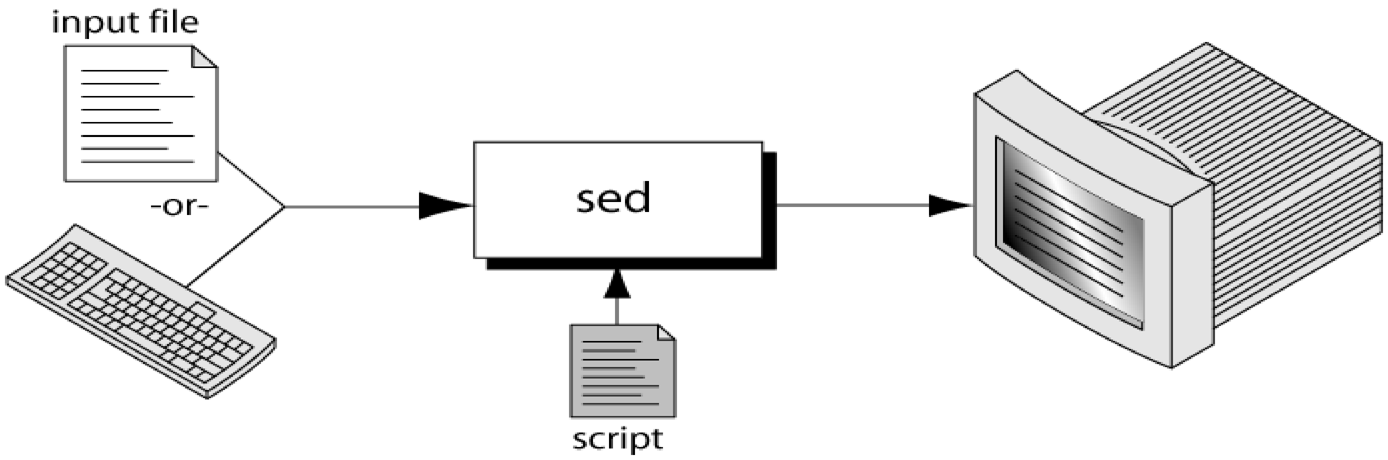
```
$  tar –tvf  dir123.tar
```

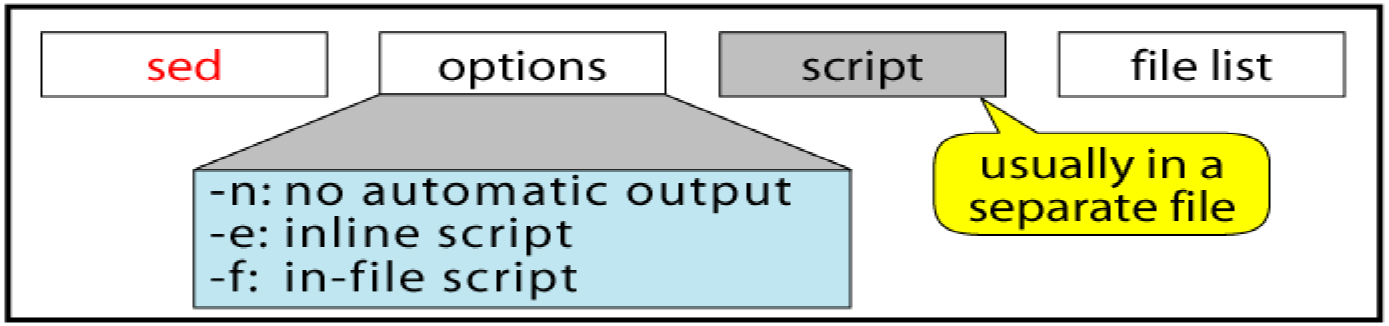
To extract tar file

```
$ tar –xvf dir123.tar
```

Sed

- Sed is one of the most powerful filters. it stands for stream editor. Despite its name editing, it does not modify the original file. It reads the standard input (a keyboard or file), processes it using a separate file called sed script and writes the standard output.





Sed

Options:

-e: it is a default option. It indicates that the script is on the command line.

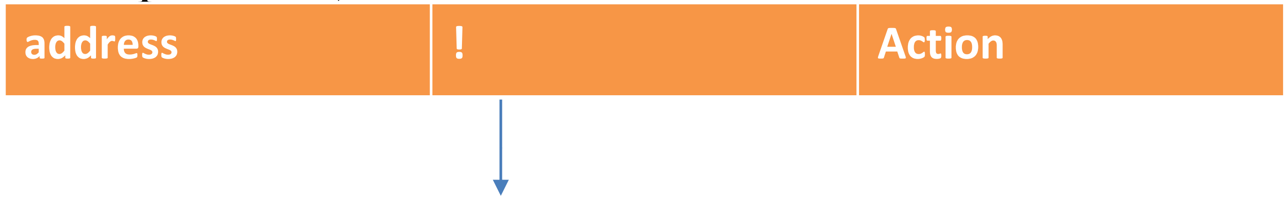
-f: it indicates that script is in a file which immediately follows this option.

-n: it suppresses the automatic output. That is, it will not display the contents of the pattern space.

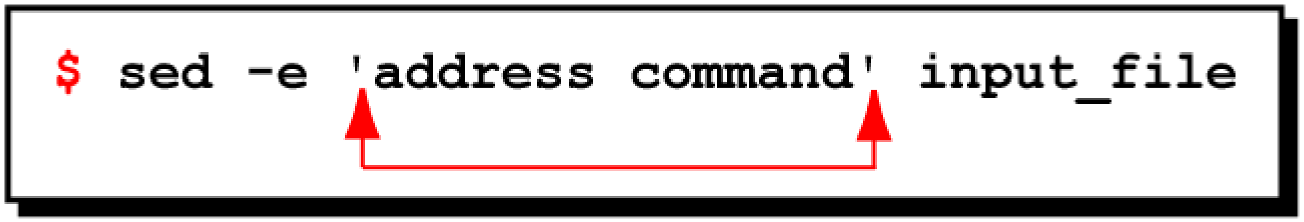
Scripts, Operation

A script is nothing more than a file of commands or instructions.

Each command consists of an address and an action, where the address can be a pattern (regular expression)



Complement operator(optional)
A Sed instruction



```
$ sed -e 'address command' input_file
```

(a) Inline Script

```
$ sed -f script.sed input_file
```

(b) Script File

Scripts

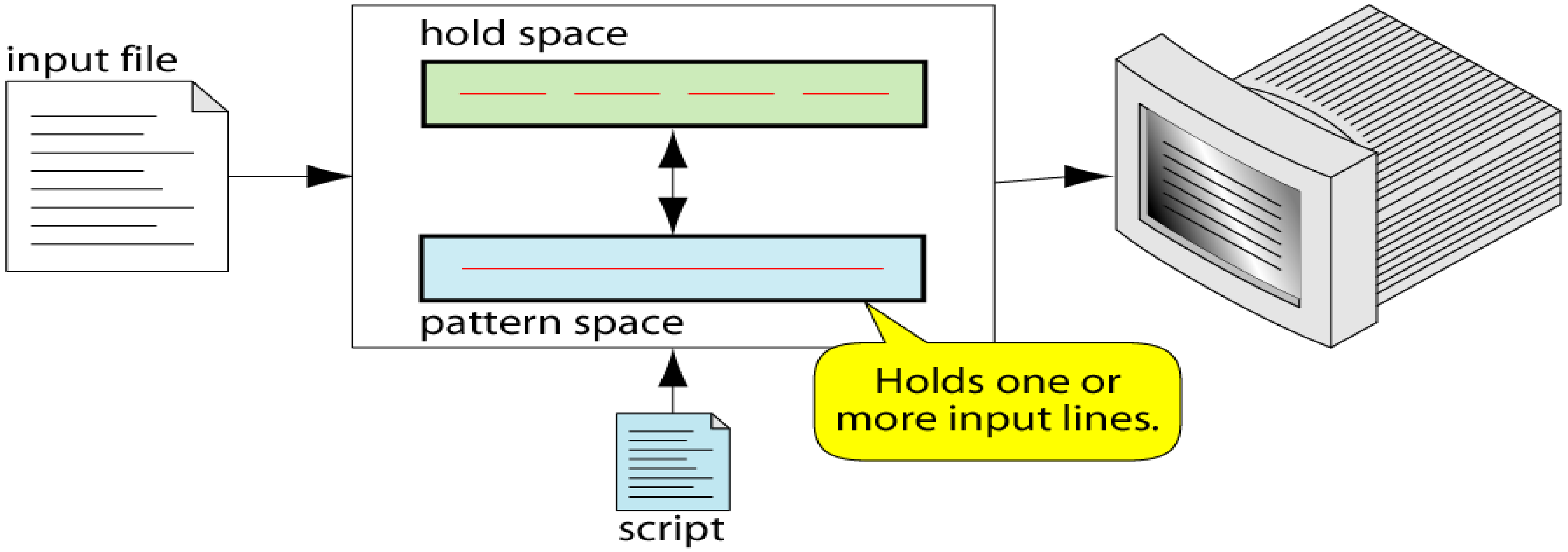
Here is an example of sed script namely sample.sed

Sample.sed

```
# this is one line comment
#this is another comment
3d
1,10s/friends/buddies/
```

In the sample script, the first two line are the comments. The third and fourth lines are the instructions. The former specifies the third line in the input file is to be deleted while the later specifies to replace all occurrences of “friends” by “buddies ” in the input lines 1 through 10.

Scripts, Operation



As each line of the input file is read, *sed* reads the first command of the script and checks the address or pattern against the current input line

If there is a match, the command is executed

If there is no match, the command is ignored

Addresses

Address determines which lines in the input file are to be processed by the command(s)

if no address is specified, then the command is applied to each input line

Address types:

- Single-Line address
- Set-of-Lines address
- Range address
- Nested address

Single-Line Address

- Specifies only one line in the input file
- special: dollar sign ($) denotes last line of input file

Addresses

Examples:

- show only line 3

sed -n -e '3 p' input-file

- show only last line

sed -n -e '$ p' input-file

- substitute —"endif" with —"fi" on line 10

sed -e '10 s/endif/fi/' input-file

Set-of-Lines Address

- use regular expression to match lines
- written between two slashes
- process only lines that match
- may match several lines
- lines may or may not be consecutives

Examples:

- /^A/command (Matches all lines that start with 'A')
- /^$/colmmand (Matches blank lines)

Addresses

Range Address

- Defines a set of consecutive lines

Format:

- start-addr,end-addr (inclusive)

following are the possible combinations

- 10,50   line-number,line-number
- 10,/R.E/   line-number,/RegExp/
- /R.E./,10 /   RegExp/,line-number
- /R.E./,/R.E/   /RegExp/,/RegExp/

Examples :

4,9   (Matches all lines 4 through 9)

2,/^A/command (starts matching from line 2 to a line that starts with A)

/^A/,15   (starts matching from a line that begins with A to 15)

/^A/,/$B/command (starts matching from a line that begins with A to a line that ends with B)

- $ sed –n ‘3-6p’ f1.txt

To print the lines other than 3-6 use !

$ sed –n ‘ 3,6!p’ f1.txt

$ sed –n ‘/[cC]omputers/p’ f1.txt

Addresses

Nested Address

- Nested address contained within another address

Example:

To delete all blank lines between line 11 and 20 the instruction is as follows

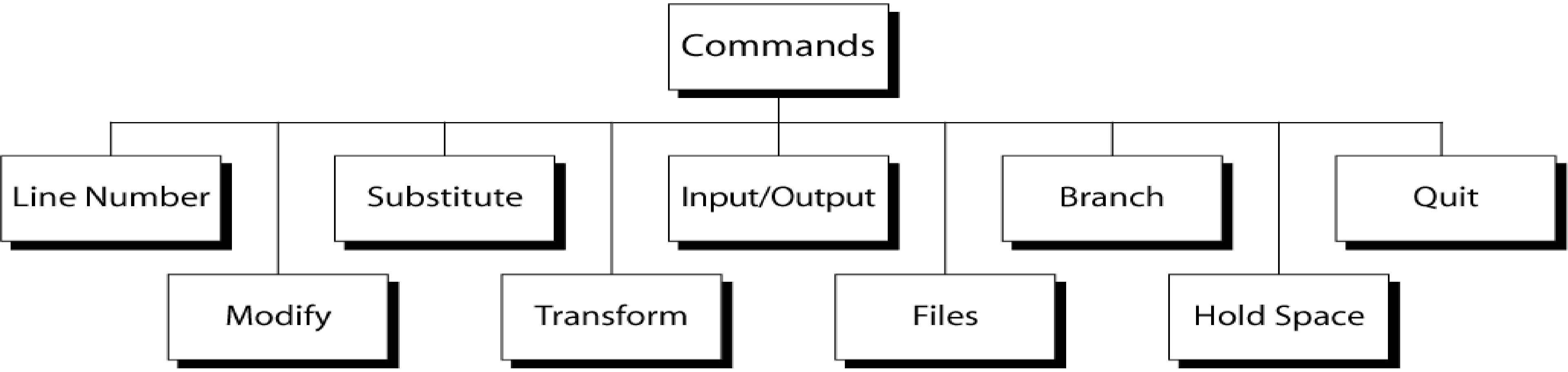
```
11,20,{
        /^$/ d
        }
```

- To delete all the lines that contain the word “Happy” as well as “life” is as follows

```
/Happy/ {
                /life/d
                }
```

- print lines that do not contain “obsolete”
- **sed -e ‘/obsolete/!p’ input-file**

Commands



Line Number

line number command (=) writes the current line number before each matched/output line

Example:

```
$ sed '=' twister.dat
```

This command print the file twister.dat with line number

✓sed ‘=’ command accepts only one address, so if you want to print line number for a range of lines, you must use the curly braces.

Syntax:

```
 # sed -n '/PATTERN/,/PATTERN/ {
=
p
}' filename
```

•Print the line numbers for the lines matches from the pattern “Oracle” to “Productivity”.

```
$ sed -n '/Oracle/,/Productivity/{
=
p
}'  f1.txt
```

o/p:

```
2
Databases - Oracle, mySQL etc.
3
Security (Firewall, Network, Online Security etc)
4
Storage in Linux
5
Productivity (Too many technologies to explore, not much time available)
```

•Find the line number which contains the pattern
•The below sed command prints the line number for which matches with the pattern “Databases”

```
$ sed -n '/Databases/=' f1..txt
```

o/p :

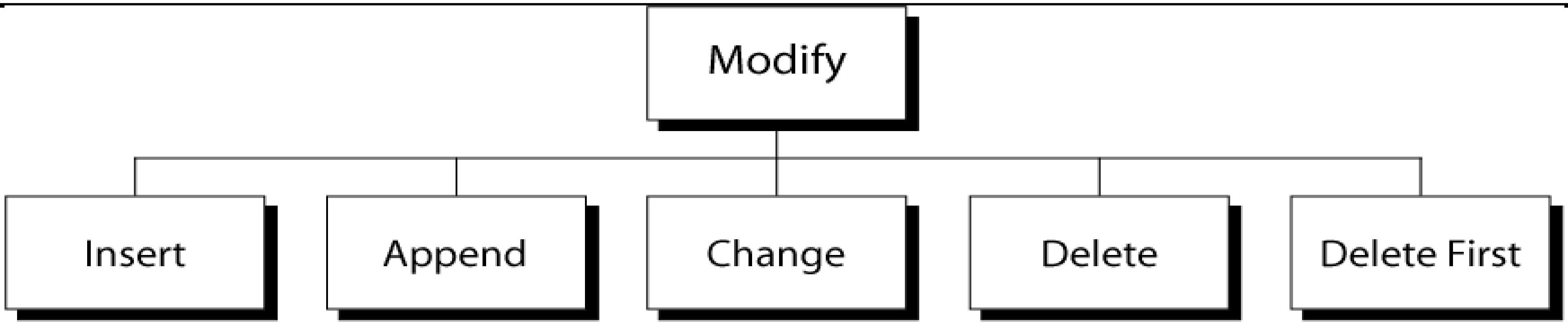
```
2
```

Print the total number of lines in a file
Line number of the last line of the file will be the total lines in a file. Pattern $ specifies the last line of the file.

```
$ sed -n '$='  f1.txt
```

```
6
```

Commands



Insert Command: I

- adds one or more lines directly to the output before the address:
- inserted “text” never appears in sed‘s pattern space
- cannot be used with a range address; can only be used with the single-line and set-of-lines address types

Syntax:

- [address] i\

      text   \

1. Add a line before the 4th line of the line:

Ex: $ cat f.txt

```
Linux Sysadmin
Databases - Oracle, mySQL etc.
Security (Firewall, Network, Online Security etc)
Storage in Linux
Productivity (Too many technologies to explore, not much time available)
Windows- Sysadmin, reboot etc.
```

$ sed '4 i\ Cool gadgets and websites'   f.txt

```
Linux Sysadmin
 Databases - Oracle, mySQL etc.
Security (Firewall, Network, Online Security etc)
Cool gadgets and websites
Storage in Linux
Productivity (Too many technologies to explore, not much time available)
Windows- Sysadmin, reboot etc.
```

2. Insert a line before every line with the pattern

```
$ sed '/Sysadmin/i \ Linux Scripting' thegeekstuff.txt
```

o/p :

```
Linux Scripting
Linux Sysadmin
 Databases - Oracle, mySQL etc.
Security (Firewall, Network, Online Security etc)
Storage in Linux
Productivity (Too many technologies to explore, not much time available)
Linux Scripting
Windows- Sysadmin, reboot etc.
```

3. Insert a line before the last line of the file.

Insert a line “Website Design” before the last line of the file.

```
$ sed '$ i\  Website Design' thegeekstuff.txt
```

o/p :

```
Linux Sysadmin Databases - Oracle, mySQL etc.
 Security (Firewall, Network, Online Security etc)
Storage in Linux
 Productivity (Too many technologies to explore, not much time available)
Website Design
Windows- Sysadmin, reboot etc.
```

Commands

Append Command: a

- adds one or more lines directly to the output after the address:
- Similar to the insert command (i), append cannot be used with a range address.
- Appended ―text‖ does not appear in sed‘s pattern space.

Syntax:

```
[address] a\
        text
```

EX:

Script.sed

```
#script to append  the text at the end of the file
$a\
Is it difficult? Try again!
```

Addresses

Change Command: c

- replaces the lines matching the address with x:new text
- accepts four address types:
- single-line, set-of-line, range, and nested addresses.

Syntax:

- **[address1[,address2]] c\**
  text

Ex: $sed –f script.sed twister.dat

Script.sed

```
5c\
Red Blood Blue Blood\
Deadly Dinosaurs Danced Dizzly
```

o/p:

```
Bad Black Bread

Fresh Fried Fish
```

Commands

Six Thick Thistle Sticks

Short Socks with Spots

Red Blood Blue Blood

Deadly Dinosaurs Danced Dizzly

I Scream ,You Scream, We all Scream for Ice Cream

Delete Command:

There are 2 versions of delete commands

1) d: it deletes all the lines in the pattern space

2) D:it deletes only the first line of the pattern space

Syntax:

[address1[,address2]] d

Commands

Ex: to delete all the lines that starts with a capital letter either B or I

```
$sed '/^[BI]/d'  twister.dat
```

o/p:

```
Fresh Fried Fish
Six Thick Thistle Sticks
Short Socks with Spots
```

Commands

Substitute command:

The substitute command, replaces the text matching the search pattern with a

replacement string.

Syntax:

- *[address(es)]s/pattern/replacement/[flags]*
  - *pattern* - search pattern
  - *replacement* - replacement string for pattern
  - *flags* - optionally any of the following
    - **n** it replaces only the specified occurrence of *pattern*
      - Here n is an integer
    - **g** it replaces all occurrences of a *pattern* in pattern space
      - By default it replaces the first occurrences of a pattern
    - **p** print contents of pattern spac
    - W it writes a file

Commands

Transform command(y):

- The Transform command (y) operates like tr, it does a one-to-one or character-to-character replacement
- Transform accepts zero, one or two addresses
- [address[,address]]y/abc/xyz/
  - every *a* within the specified address(es) is transformed to an *x*. The same is true for *b* to *y* and *c* to *z*
  - y/abcdefghijklmnopqrstuvwxyz/ABCDEFGHIJKLMNOPQRSTUVWXYZ/ changes all lower case characters on the addressed line to upper case

Commands

sed i/o commands :

There are 5 i/o commands

1.Next 2.Append next 3.Print

4.Print first line 5.List

1.Next:

The next command, n reads the next line from the input file into the pattern space. before reading the next line , it sends the current contents of the pattern space to the standard output, deletes the current line in it and then copies the next line

Ex:the command to delete a blank line if it appears after a line that starts with a digit is given below

```
$sed –f myscript.sed file1.dat
```

Commands

Myscript.sed

```
#script to demonstrate the next command
/^[0-9]/{
            n
            /^$/d
            }
```

2 .Append Next(N):

- The append next command, adds the next input line to the current contents of the pattern space
- This command is useful when applying patterns to two or more lines at the same time.
- Ex : the command to append next line to the current line if the current line ends with a colon(:)

Commands

Script.sed

```
/:$/N
#replace the new line with a single space
s/\n/ /
```

3.Print(p)

The Print command (**p**) prints the current content of the pattern space to standard otput, useful if the ***-n*** option has been specified

- Syntax: **[address1[,address2]]p**
- Note: if the ***–n*** option has not been specified, **p** will cause the line to be output twice!
- Examples:

$sed –n ‘1,5p’ file1.dat   it prints the lines from 1 to 5.

/^$/,$p will display the lines from the first blank line through the last

Commands

4.Print first line(P)

It prints only the first line of the pattern space. it stops print when it finds a new line character.

List Command

5.List(l)

The list command, l converts the unprintable characters into printable characters. it prints the octal value of the unprintable characters where each octal value is preceded by'\' character

Ex:

```
$sed –n '|' myfile.dat
```

Myile.dat
This is a tab and new line
Another tab

Commands

Out put:

```
This is a tab\t and newline$
\t another tab.$
```

File commands

- allows to read and write from/to file while processing standard input
- read: r command
- write: w command

Read File command

Syntax: **r filename**

Ex: write the command to insert the content of file2.txt before the contents of file1.txt

```
$ sed ‘ 1r file2.txt ’ file1.txt
```

Commands

Ex: write the command to save first 20 lines from file1.txt to a file output.txt

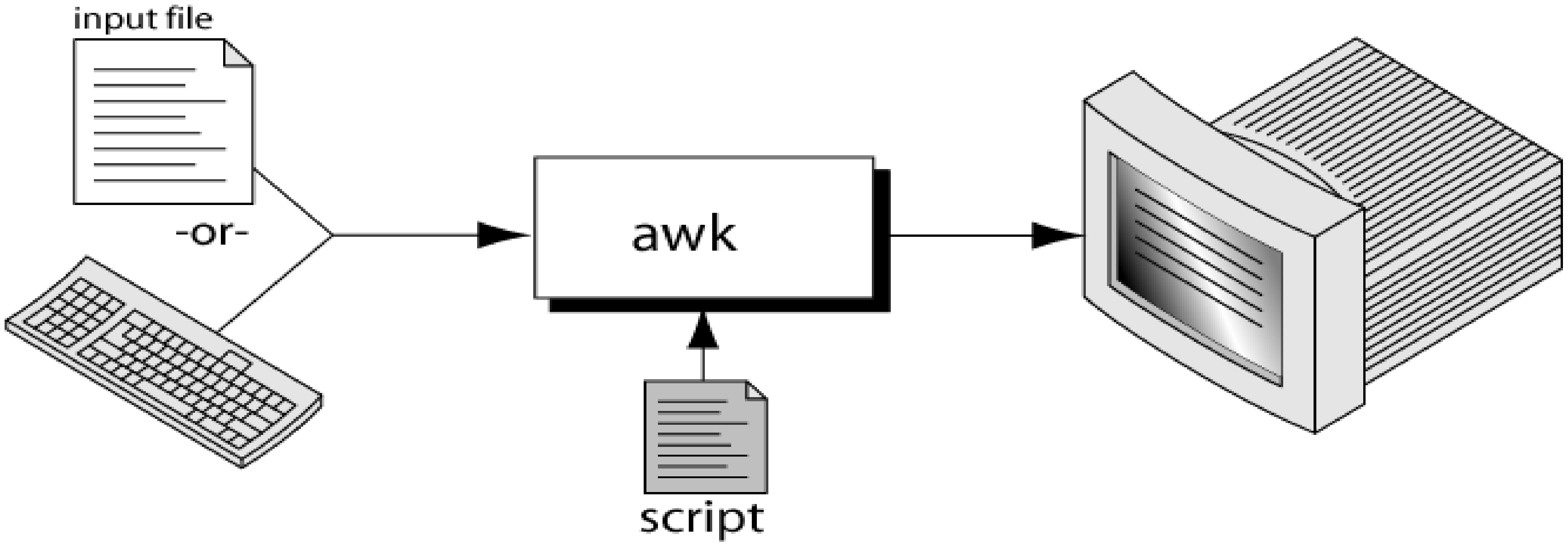
```
$ sed –n1,20w output.txt file1.txt
```

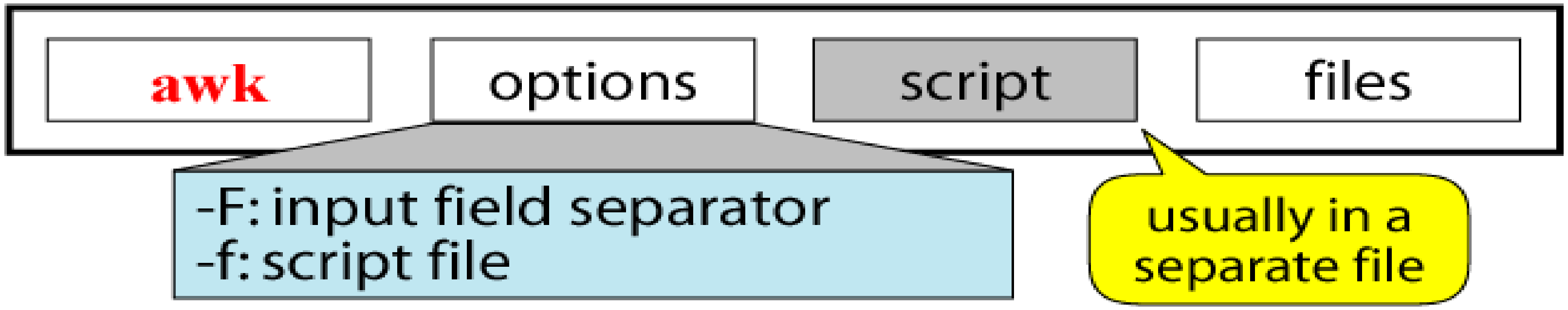
Quit

- Quit causes **sed** to stop reading new input lines and stop sending them to standard output
- It takes at most a single line address
  - Once a line matching the address is reached, the script will be terminated
  - This can be used to save time when you only want to process some portion of the beginning of a file
- Example: to print the first 100 lines of a file (like *head*) use:
  - **sed '100q' filename**
  - sed will, by default, send the first 100 lines of *filename* to standard output and then quit processing

Awk

The awk utility, which takes its name from the initials of its authors (Aho, Weinberger, and Kernighan), is a powerful programming language.





Execution

In addition to input data, awk also requires one or more instructions that provide editing instructions.

A few instructions can be entered at the command line from the keyboard.

Most of the time, the instructions are placed in a file known as an awk script.

Syntax

```
$awk options awk scripts file(s)
```

Options

-f: it indicates that script is in a file which immediately follows this option.

-F:this option specifies the input field seperator. By default, it is space or a tab.

Ex:

```
$awk –F “\t” –f firstscript.awk info.dat
```

Fields and Records

The awk utility views a file as a collection of fields and records.

A field is a unit of data that has informational content.

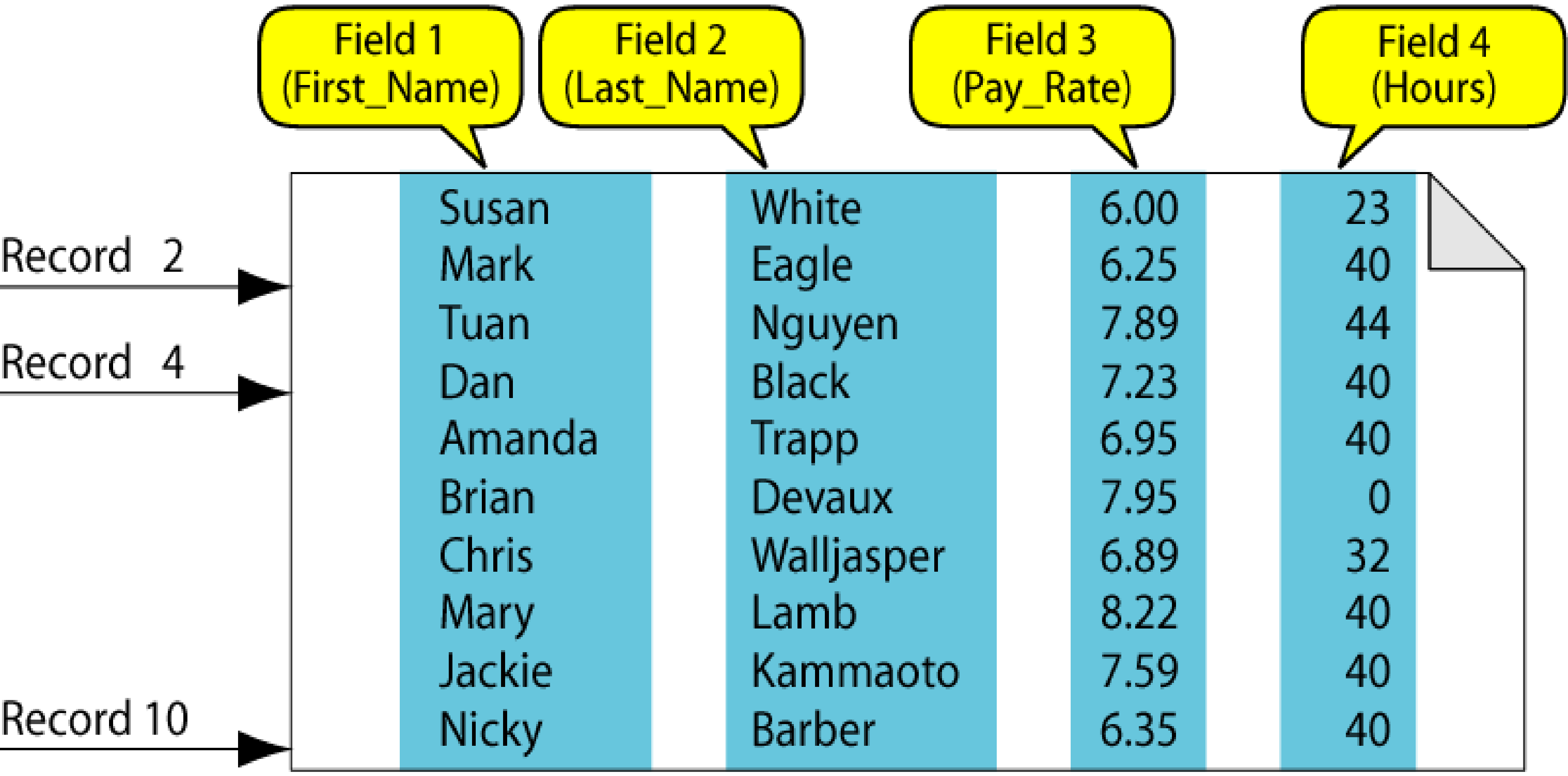
Each field of information is separated by one or more whitespace characters or other separator.

Each line is a record. A record is a collection of fields.

When a file is organized into records, we call it a data file, as contrasted with a text file.

awk also can handle a text file.

A Data File



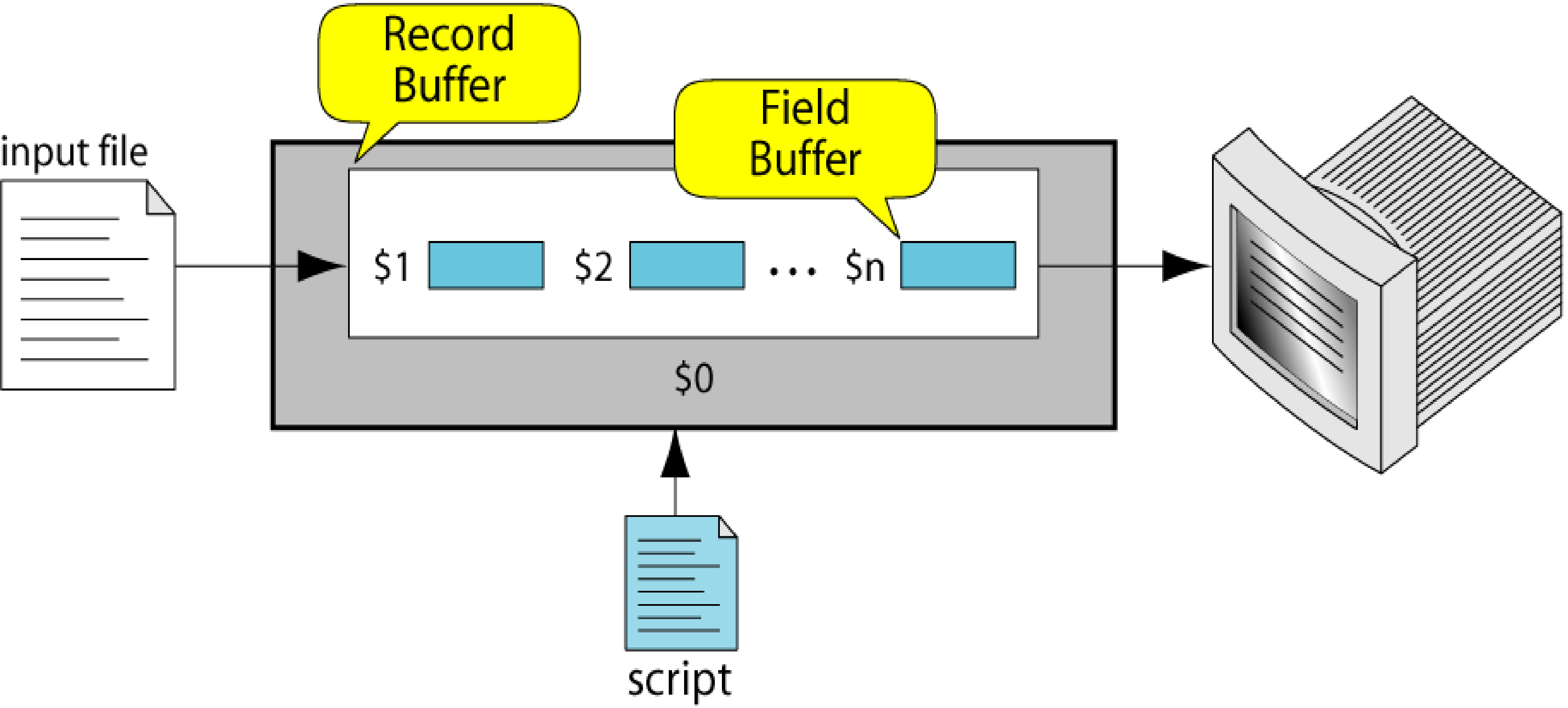
Susan	White	6.00	23
Mark	Eagle	6.25	40
Tuan	Nguyen	7.89	44
Dan	Black	7.23	40
Amanda	Trapp	6.95	40
Brian	Devaux	7.95	0
Chris	Walljasper	6.89	32
Mary	Lamb	8.22	40
Jackie	Kammaoto	7.59	40
Nicky	Barber	6.35	40

A file with 10 records, each with four fields

Field Buffers

There are as many field buffers available as there are fields in the current record of the input file.

Each buffer has a name, which is the dollar sign $ followed by the field number in the current record.



Record Buffer

There is only one record buffer available. Its name is $0.

It holds the whole record.

System Variables

FS	Input field separator
RS	Input record separator
OFS	Output field separator
ORS	Output record separator
NF	Number of nonempty fields in a record
NR	Number of records read from all files
FNR	Number of records in current file
FILENAME	Name of the current file

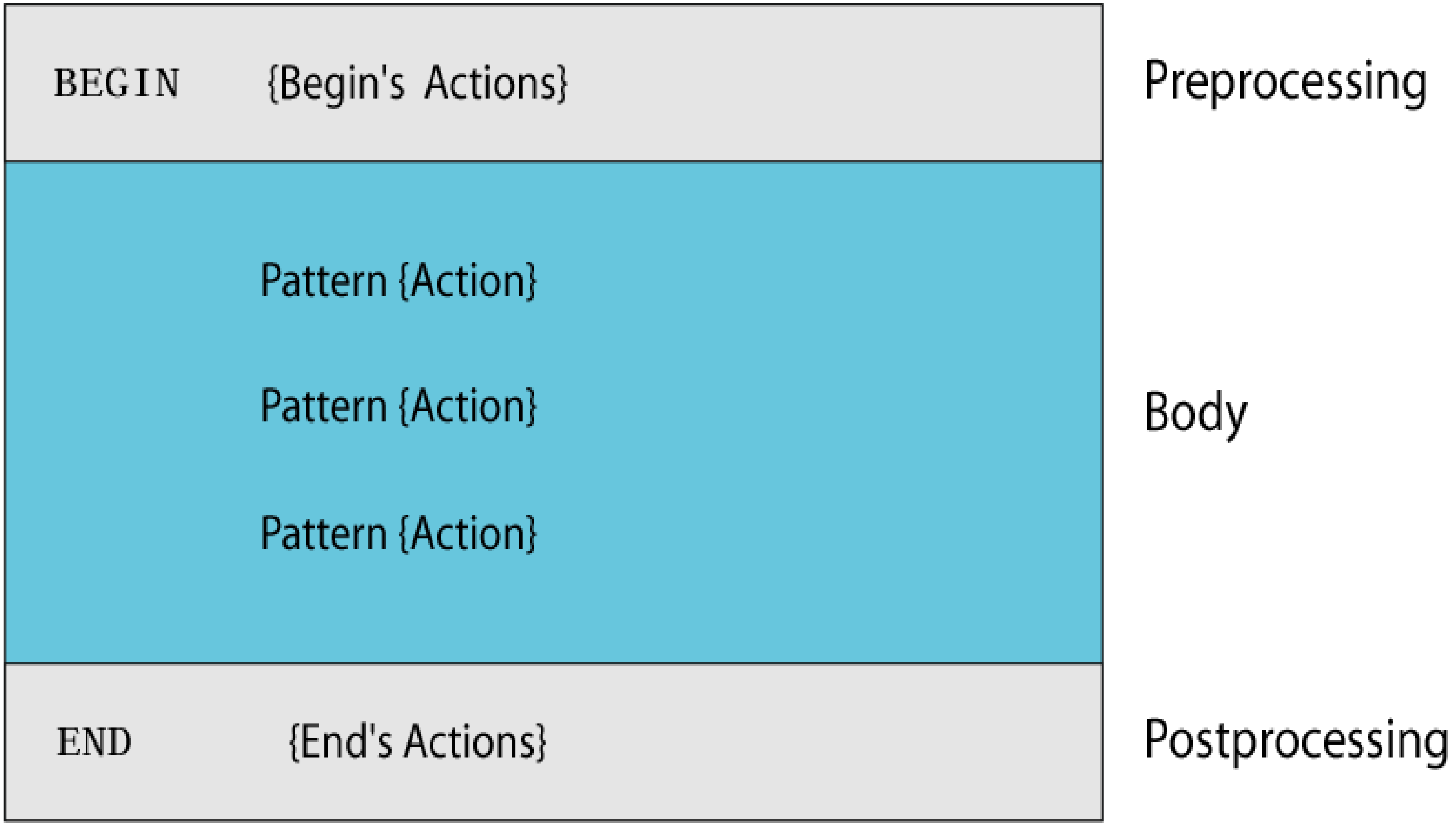
System Variables

Examples :

```
% cat emps
Tom Jones       4424    5/12/66 543354
Mary Adams       5346    11/4/63 28765
Sally Chang     1654    7/22/54 650000
Billy Black     1683    9/23/44 336500

% awk '{print NR, $0}' emps
1 Tom Jones     4424    5/12/66 543354
2 Mary Adams    5346    11/4/63 28765
3 Sally Chang   1654    7/22/54 650000
4 Billy Black   1683    9/23/44 336500
```

awk Script Design



awk Script

- BEGIN: pre-processing
  - performs processing that must be completed before the file processing starts (i.e., before awk starts reading records from the input file)
  - useful for initialization tasks such as to initialize variables and to create report headings
- BODY: Processing
  - contains main processing logic to be applied to input records
  - like a loop that processes input data one record at a time:
    - if a file contains 100 records, the body will be executed 100 times, one for each record

awk Script

- END: post-processing
  - contains logic to be executed after all input data have been processed
  - logic such as printing report grand total should be performed in this part of the script

Operations

- The awk utility reads an input file, line by line and applies all the instructions in awk script to each input line. It does not print the lines unless the print action is specified .consider a records .dat with five fields i.e,s.no,name,test1,test2,test3 and four records.

1	A	75	81	89
2	B	86	94	82
3	C	90	71	96
4	D	82	70	85

```
$awk –f total.awk records.dat

Total.awk

#initialization section

BEGIN {print “name\t total”}
```

Operations

```
#body section
{total=$3+$4+$5}
{print $2+”\t” total}
#END of job
END {print “Best Result”}
```

Output:

```
Name    total
A       245
B       262
C       257
D       237
Best Result
```

Pattern/Action

```
pattern {statement}
```

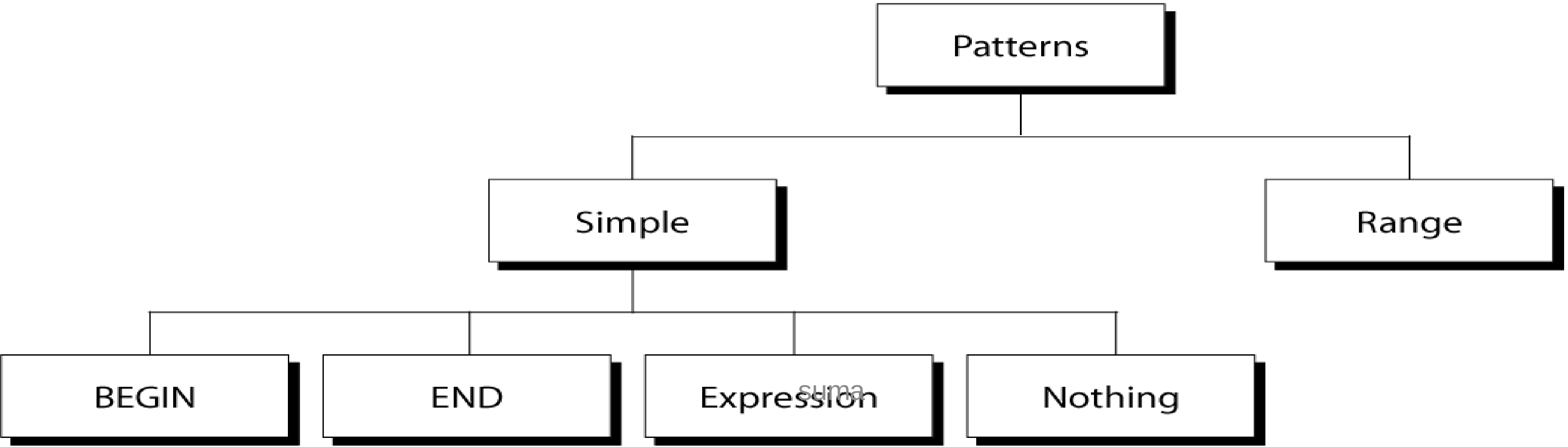
(a) One Statement Action

```
pattern {statement1; statement2; statement3}
```

(b) Multiple Statements Separated by Semicolons

```
pattern
{
     statement1
     statement2
     statement3
}
```

(c) Multiple Statements Separated by Newlines



suma

Expression Pattern types

- match
  - entire input record

    regular expression enclosed by ‘/’s
  - explicit pattern-matching expressions

    ~ (match), !~ (not match)

- expression operators
  - arithmetic
  - relational
  - logical

Example: match input record

```
% cat employees2
Tom Jones:4424:5/12/66:543354
Mary Adams:5346:11/4/63:28765
Sally Chang:1654:7/22/54:650000
Billy Black:1683:9/23/44:336500

% awk –F: '/00$/' employees2
Sally Chang:1654:7/22/54:650000
Billy Black:1683:9/23/44:336500
```

Arithmetic Operators

Operator	Meaning	Example
+	Add	x + y
-	Subtract	x – y
*	Multiply	x * y
/	Divide	x / y
%	Modulus	x % y
^	Exponential	x ^ y

Example:

```
% awk '$3 * $4 > 500 {print $0}' file
```

Relational Operators

Operator	Meaning	Example
<	Less than	x < y
< =	Less than or equal	x < = y
==	Equal to	x == y
!=	Not equal to	x != y
>	Greater than	x > y
> =	Greater than or equal to	x > = y
~	Matched by reg exp	x ~ /y/
!~	Not matched by req exp	x !~ /y/

Logical Operators

Operator	Meaning	Example
&&	Logical AND	a && b
||	Logical OR	a || b
!	NOT	! a

Examples:

```
% awk '($2 > 5) && ($2 <= 15)
                {print $0}' file
% awk '$3 == 100 || $4 > 50' file
```

Logical Operators

Operator	Meaning	Example
&&	Logical AND	a && b
||	Logical OR	a || b
!	NOT	! a

Examples:

```
% awk '($2 > 5) && ($2 <= 15)
                 {print $0}' file
% awk '$3 == 100 || $4 > 50' file
```

Range Patterns

Matches ranges of consecutive input lines

<u>Syntax:</u>

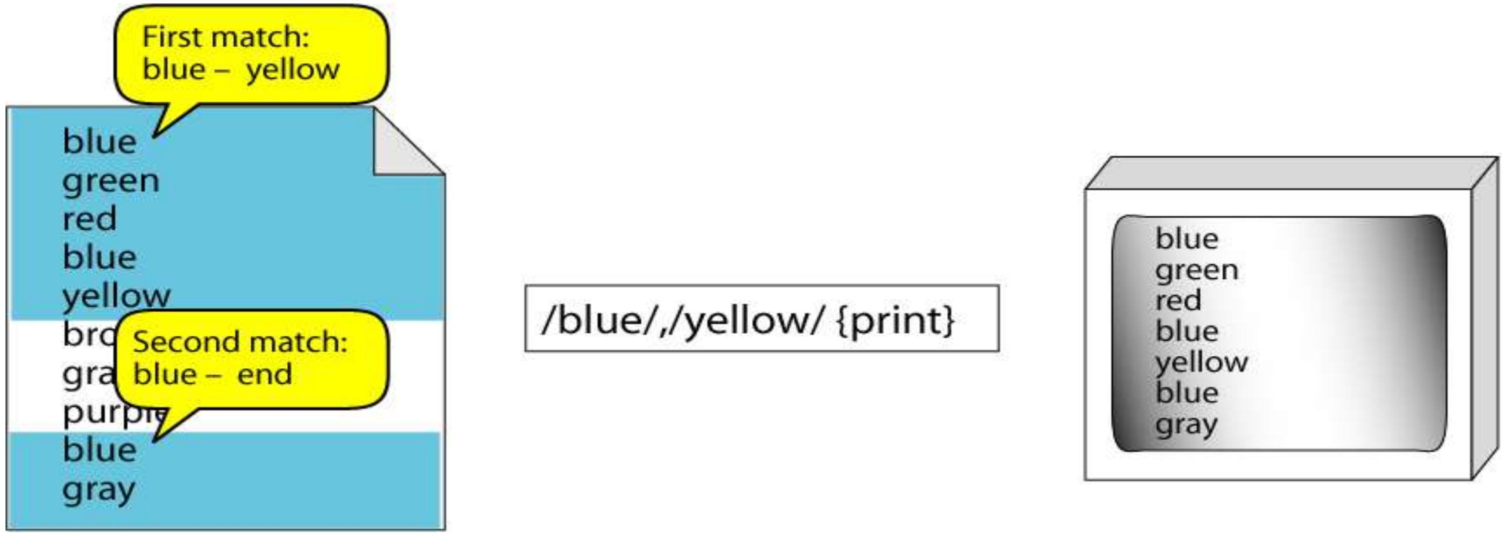
    pattern1 , pattern2 {action}

pattern can be any simple pattern

pattern1 turns action on

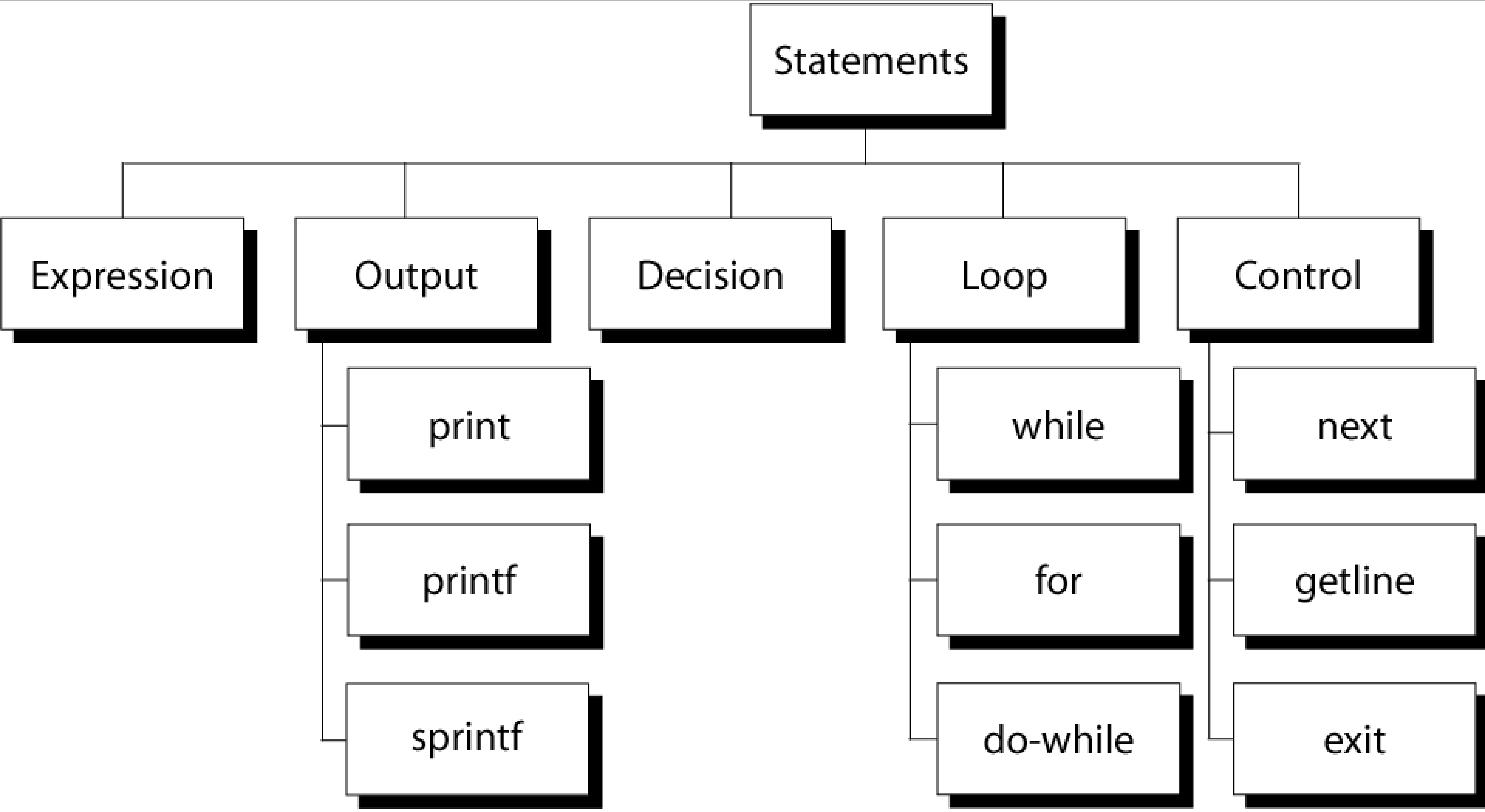
pattern2 turns action off

Example:



```
/blue/,/yellow/ {print}
```

awk Actions


Statements
Expression
Output
Decision
Loop
Control
print
printf
sprintf
while
for
do-while
next
getline
exit

awk expressions

Expression is evaluated and returns value

- consists of any combination of numeric and string constants, variables, operators, functions, and regular expressions

Can involve variables

- As part of expression evaluation
- As target of assignment

awk variables

- A user can define any number of variables within an awk script
- The variables can be numbers, strings, or arrays
- Variable names start with a letter, followed by letters, digits, and underscore
- Variables come into existence the first time they are referenced; therefore, they do not need to be declared before use
- All variables are initially created as strings and initialized to a null string “”

Format:

```
    variable = expression
```

Examples:

```
% awk '$1 ~ /Tom/
        {wage = $3 * $4; print wage}'
      filename
% awk '$4 == "CA"
        {$4 = "California"; print $0}'
      filename
```

awk assignment operators

=	assign result of right-hand-side expression to left-hand-side variable
++	Add 1 to variable
--	Subtract 1 from variable
+=	Assign result of addition
-=	Assign result of subtraction
*=	Assign result of multiplication
/=	Assign result of division
%=	Assign result of modulo
^=	Assign result of exponentiation

Output Statements

print

  print easy and simple output

printf

  print formatted (similar to C printf)

sprintf

  format string (similar to C sprintf)

Awk control structures

- Conditional
  - if-else
- Repetition
  - for
    - with counter
    - with array index
  - while
  - do-while

  - also: break, continue

if Statement

Syntax:

```
if (conditional expression)
    statement-1
else
    statement-2
```

Example:

```
if ( NR < 3 )
    print $2
else
    print $3
```

for Loop

<u>Syntax:</u>

```
for (initialization; limit-test; update)
      statement
```

<u>Example:</u>

```
for (i = 1; i <= NR; i++)
{
     total += $i
     count++
}
```

- For loop for arrays

<u>Syntax:</u>

```
for (var in array)
      statement
```

<u>Example:</u>

```
for (x in deptSales)
{
     print x, deptSales[x] }
```

while Loop

Syntax:

```
while (logical expression)
        statement
```

Example:

```
i = 1
while (i <= NF)
{
       print i, $i
       i++
}
```

do-while Loop

Syntax:

```
do
        statement
while (condition)
```

- statement is executed at least once, even if condition is false at the beginning

Example:

```
i = 1
do {
  print $0
  i++
} while (i <= 10)
```