UNIX Commands

To Create a File:

Syntax: $ cat [option] > <filename>

```
Ex: $ cat > file1
Hello..
This is my first File
Have a Nice Day
Bye
```

Ctrl+d (Save)

Ctrl+d – Possible Completer
Ctrl+c – Cancel foreground Job
Ctrl+z – Stop (interrupted) a foreground Job

To View a already existing File:

Syntax: $ cat <filename>

```
Ex: $ cat file1
```

To append data to an existing file

Syntax: $ cat >> <filename>

```
Ex: $ cat >> file1
```

To Create a Multiple file with help of cat command

Syntax: $ cat <filename1 filename2 ...filename (n) >

```
Ex: $ cat >f1 >f2 >f3 >f4
Hello....
This is file number f4
We create multiple files
^d (Save)
```

Display the record number to the particular file

Syntax: $ cat [option] <filename>

```
Ex: $ cat -n file1
```

To ignore the blank Records

Syntax: $ cat [option] <filename>

```
Ex: $ cat -b file1
```

Create a Hidden file

Syntax: $ cat [option] <filename>
```
Ex: $ cat >.file1
```
Create a Directory

Syntax: $ mkdir [option] <Directory name>
```
Ex: $mkdir raju
```

Create Multiple Directories

Syntax: $ mkdir [option] <Directory names>
```
Ex: $ mkdir d1 d2 d3 d4 d5
```

Create Multi – level Directories

Syntax: $ mkdir [option] <Directory names>
```
Ex: $ mkdir –p /d5/d51/d52/d53
```

Create Multiple Sub directories

Syntax: $ mkdir [option] <Directory names>
```
Ex: $ mkdir –p d3/d31 d4/d41
```

To Display present working Directory

Syntax: $ pwd
```
Ex: $ pwd
```

To Change the Directory
Syntax: $ cd <Directory name>
```
Ex: $ cd raju
```

To change the directory forward
Syntax: $ cd..
```
Ex: $ cd..
```

To Move parent Directory (root)
Syntax: $ cd /
```
Ex: $ cd /
```

To Create a Hidden Directory

Syntax: $ mkdir [option] <Directory name>
```
Ex: $ mkdir .venkat
```

REMOVE COMMANDS:

To Remove a file

Syntax; $ rm [option] <filename>
```
Ex: $ rm kiran
```

To Remove Multiple Files

Syntax: $ rm [option] <filenames >
```
Ex: $ rm f1 f2 f3 f4
```

To remove files Forcibly

Syntax: $ rm [option] <filename>
```
Ex: $ rm -f kiran
```

To Remove Interactive Mode

Syntax: $ rm [option] <filename>
```
Ex: $ rm -i kiran
```

To Remove Directory

Syntax: $ rm [option] <Directory Name>
```
Ex: $ rm sagar
```

To Remove all Directories and subdirectories

Syntax: $ rm [option] <Directory Name>
```
Ex: $ rm -r sagar
```

To Remove Directories Forcibly

Syntax; $ rm [option] <Directory Name>
```
Ex: $ rm -rf sagar
Ex: $ rm -rf sagar pavan amar
```

To Remove Directories Interactively

Syntax; $ rm [option] <Directory Name>
```
Ex: $ rm -i sagar
Ex: $ rm -i sagar pavan amar
```

LIST COMMANDS

$ ls : It is a command to list the files and directories in the present working Directory

$ ls - a : It is a command to display all files and Directories including hidden files and Directories

$ls * : List information about the Files (the current directory by default). Sort entries alphabetically

$ls ~ : It list the all Backup files

$ls @ : It list the all linked files and Directories

$ls -d : It Displays the present working Directory.

$ls -i : It Displays the inode numbers of files and Directories

$ls -s : It Displays the sizes in blocks (Files & Directories)

$ls -l : It Displays the long listing files and directories in present working directory

Listing directory contents:

$ ls list a directory
$ ls -l list a directory in long (detailed) format

for example:

$ ls -l

```
drwxr-xr-x    4 vijay    user        1024 Jun 18 09:40 WAITRON_EARNINGS
-rw-r--r--    1 kiran    user      767392 Jun  6 14:28 scanlib.tar.gz
^ ^  ^  ^     ^   ^       ^           ^      ^    ^      ^
| |  |  |     |   |       |           |      |    |      |
| |  |  |     | owner   group       size   date  time    name
| |  |  |     number of links to file or directory contents
| |  |  permissions for world(others)
| |  permissions for members of group
| permissions for owner of file: r = read, w = write, x = execute -=no
permission
```

type of file: - = normal file, d=directory, l = symbolic link, and others...

ls -ld * List all the file and directory names in the current directory using long format. Without the "d" option, ls would list the contents of any sub-directory of the current. With the "d" option, ls just lists them like regular files.

$ ls -al : It Displays including hidden and log listing files and Directories

$ ls -m : It Displays all files and Directories with separated by comma (,)

$ ls -ls : It Displays all long listing Directories

$ ls --full-time : It Displays files and Directories with total information date and time

$ ls -nl : It Displays the long listing files and Directories according to modification Time

$ ls -rtl : It Displays the file and Directories with reverse order

$ ls -R : It Displays the all files and Directories Regressively (order by order)

$ ls -l : It Displays the files and Directories in a single column (vertical)

$ ls -x : It Displays the files and Directories with multiple columns

COPY COMMANDS

Syntax: $cp [option] <source file> <Destination file>

1. File to a File
2. File to a Dictionary
3. Directory to Directory

File to File

Syntax : $cp <source file> <Destination file>

```
Ex : $cp venkat amar
..... New file

Ex : $cp venkat john
      Existing file
```

Note: The content of john file overwrite with venkat

```
Ex ; $cp -f venkat kumar
```

Note : It is copy the file forcibly venkat to kumar without any permission

```
Ex : $cp -i venkat gandhi
```

Note : It is copy the file interactive mode venkat to kumar with permissions

2. File to Directory

Syntax ; $ cp <source file> <Destination Directory>

```
Ex ; $ cp venkat ajay
.... ajay is Directory

Ex : $ cp -b venkat satya
```

Note : It copys the venkat file to Satya Directory with backup

```
Ex : $ cp -bf venkat kiran
```

Note ; It copys the venkat file to Kiran Directory with forcibly backup

```
Ex : $ cp -ibf venkat yusuf
```

Note : It copys the venkat file to Yusuf Directory with backup interactively and forcibly

```
Ex : $ cp file1 file2 file3 Directory
```

Note : It copies the no of file to Directory

3. Directory to Directory

Syntax : $ cp <Source Directory> <Destination Directory>

```
Ex : $ cp Dravid Ganguly

Ex : $ cp -r Mody Venkat
```

Note : It copies the Directory to Directory (including all files and Directories)

```
Ex : $ cp -ir <Directory 1> <Directory 2>
```

Note : It copies the Directory to Directory (including with all files and directories with interactive mode)

```
Ex : $ cp -rf <Dir 1 > <Dir 2> <Dir 3> <Destination
                                                Directory>
```

Note : It copies the multiple Directories with forcibly and recessively mode

MOVE COMMANDS

This command is used to move the files and directories one place to another place

Syntax: $ mv [option] <Source file/Directory> <Target file/Directory>

These are basically 3 types
1. File to File
2. File to Directory
3. Directory to Directory

1. File to File

Syntax: $ mv [option] <Source file> <Target file>

```
Ex: $ mv file1 file2
```

```
Ex: $mv -b file1 file2
```

Note: The file1 move to file2 with backup

```
Ex: $ mv -f file1 file2
```

Note: The file1 move to file2 with forcibly

```
Ex: $ mv -if file1 file2
```

Note: The file1 move to file2 with interactive and forcibly mode

2. File to Directory

Syntax: $ mv [option] <Source file> <Target Directory>

```
Ex: $ mv file1 Dir1
```

Note: The file1 moves to Directory (Dir1)

```
Ex: $ mv -b file1 Dir1
```

Note: The file1 move to Dir1 with backup mode

```
Ex: $mv -f file1 Dir1
```

Note: The file1 move to Dir1 with forcibly mode

```
Ex: $mv -if file1 Dir1
```
Note: The file1 move to Dir1 with interactive and forcibly mode.

3. Directory to Directory

Syntax: $ mv [option] <Source Directory> <Target Directory>
```
Ex: $ mv Dir1 Dir2
```
Note: The content of Dir1 moves to Dir2

```
Ex: $ mv -b Dir1 Dir2
```
Note: The contents of Dir1 move to Dir2 with backup

```
Ex: $ mv -f Dir1 Dir2
```
Note: The contents of Dir1 moves to Dir2 with forcibly

```
Ex: $ mv -if Dir1 Dir2
```
Note: The contents of Dir1 moves to Dir2 with interactive and forcibly mode.

PATH IN UNIX

1. Absolute path
2. Relative path

1. Absolute path: It is an independent on present working directory
2. Relative path: It is a path mention from root directory

Absolute path Example:

```
Ex: $ cd /b1/b2/b3/b4/b5
  : $ cp /usr/paramesh/a1/a2/a3/file <Directory>
```

Relative Path Example:

```
Ex: $cd /b1/b2/b3/b4/b5
  : $mv ../../../../../c1/c2/c3/c4/c5
```

FILE ACCESS PERMISSIONS (FAP)

-rwx------
drwxr--r--
-rwx-w--w-

d - Directory
w - Wrire
r - Read
x - Executable
Note: Without write permissions we can't copy, remove, modify, move, crate

drwxrwxrwx
7 7 7

Directory – 777
File – 666
Default Directory Permissions – 755
Default File Permissions – 644

Read – 4
Write – 2
Executable – 1

Total Permissions 7

TO CHANGE AND MODIFY THE PERMISSIONS

1. Numeric Method
2. Symbolic Method

1. Numeric Method:

Syntax: $ chmod [permissions] <file/Directory>

```
Ex: $ chmod 700 Dir2
    $ chmod 640 file1
```

To check the changed permission use the command: ls - l

To change the permissions and check the permission using following command

```
Ex: $ chmod -c 750 Dir2
```

To change the permission with Directories and subdirectories with regressively

```
Ex: $ chmod -R 755 Dir2
```

To Change the present working Directory Permissions

```
Ex: $ chmod - R 750
```

Note : Plz don't' try this command on the system

2. Symbolic Method

User – u
Group – g
Others – o
All – a
Read – r
Write – w
Execute – x

Syntax: $ chmod [options] <File/Directory>

To Add Write permission to Group and Others
```
Ex: $ chmod g+w, o+w Dir1
```

To Remove the Write Permission to User
```
Ex: $ chmod u-w Dir1
```

To Remove the Write and execute permissions to Group and Others
```
Ex: $chmod g-r, g-x, o-r, o-x Dir1
```

Append the Write permissions to all
```
Ex: $chmod u+w, g+w, o+w Dir1
            (or)
     $chmod a+w Dir1
```

UMASK (User File Creation Mask)
- cuting, remove, hidden
By Default umask value is 022

Directory	File
777	666
022	022
755	644

WILD CARDS

^ - Carrot or Cap
? - Single Character
* - Multiple Characters
[ ] - Range of Characters

```
Ex: ls ? - This command is used list with single character
           files
```

```
ls ?? - This command is used to list with double character
        files

ls ? [0-9] - This command is used to list the files starting
             with alphabetically and with numeric

ls * - This command is used to display the all the files and
       directories and files Subdirectories

ls a* - This command is used to display the files with
        starting letter with 'a' Character
```

Remove Commands

Syntax: rm [option] <file/directory name>

```
Ex: rm * - This command is used to delete all the files in
current directories
Note: plz don't use this command without permission

Ex: rm ?  - To delete single character files
Ex: rm ?? - To delete double character files
Ex: rm a* - To delete the files starting with
            character 'a'
```

Links

Call the link function to create a link to a file.
Call the link function to create a link named FILE2 to an existing FILE1

1. Hard link
2. Soft Link

1. **Hard Link** : Hard link can be build single file system
   Hard link can recognized it links same permissions and same size, same inon number
   If the data is source loss we got from hard link
   inon number is given space allocated
   inon numbers can recognized different logical names

Syntax : ln <source file> <target file>

```
Ex : ln -f vankat kumar ( f – forcibly)
Ex : ln -i venkat kumar (i – interactively)
```

2. **Soft Link** : It can be built across the file system
   It the source file is delete we can't retrieve from target file

Syntax : ln -s <source file> <target file>

```
Ex : ln sd Dir1 Dir2 (d – Directories)
Ex : ln -sf File1 File2 (f – forcibly)
Ex : ln -si File1 File2 (s – inode number)
```

FIND COMMANDS

It search for files in a directory hierarchy

Syntax: find [path...] [expression]
/ - root
~ - Home Directory
. - Present working directory

```
Ex: $ find /bin –type f
It searches the files and directories absolute path of the
root.

Ex: $ find ~ -type d
It searches the directories and subdirectories in the Home
directory

Ex: $ find ~ -type l
It searches the linked files

Ex: $ find ~ -type f -name “file1”
It searches the absolute path of the filename

Ex: $ find ~ -type d -name “raj”
It searches the absolute path of the Directories

Ex: $ find ~ -type f -name “file1” exce cat{}\;
It searches the absolute path of the filename and displays
the contents of file.

Ex: $ find ~ -type f -name “file1” exce cat {}\ exce rm{}\;
It searches the absolute path of the filename and displays
the content of file, and removes at the same time.

Ex: $ find ~ -type f -perm 644
```

```
It searches the file under with 644 permission

Ex: $ find ~ -type f -perm 644 -exce chmod 640 {}\;
It searches the filename under with permission 644 and change
the file permissions to 640.

Ex: $ find ~ -type f size 0;
It Searches the Zero byte files
```

Linux: 512 bytes – 1 block

```
Ex: $ find ~ -type f size 100c (Exactly 100 bytes file)
    $ find ~ -type f size +100c (above 100 bytes file)
    $ find ~ -type f size -100c (below 100 bytes file)

Ex: $ find ~ -type d size 8b
It searches the 8bytes Directories

Ex: $ find ~ -type f -i num "1098"
It searches the file with inode number

Ex: $ find ~ type d – inum "1024"
It searches the directories with inode number
```

Time

-a – Access
-c – Changed
-m – Modified

```
Ex: $ find ~ -type f –a min 30
It searches the file before 30 min access

Ex: $ find ~ -type f –a time –n/n/+n
It searches the file before few days
```

Vi Editor (Visual Editor)

Vi is a text editor that is upwards compatible to Vi. It can be used to edit all kinds of plain text. It is especially useful for editing programs.

1. Esc mode
2. Insert mode
3. Colon mode

```
A, a
```

I, i
O, o
S With the help of these keys move esc mode to inert mode

Shift: with the help of shift: move to Esc mode to colon mode

Esc Mode Commands k h l j

By press
'L' - The cursor moves right direction
'h' - The cursor moves one character left
'k' - The cursor moves one character up
'j' - The cursor moves one character down

'w' - The cursor moves next line first
5w   - The cursor moves 5 words forward
nw   - The cursor moves n words forward
b   - The cursor moves previous word
5b   - The cursor moves 5 words before
nb   - The cursor moves n words before

e   - The cursor moves next word last character
$ - The cursor moves the end of the record
^   - The cursor moves the starting of the record

gg   - The cursor moves the first record
15gg - The cursor moves the 15th record of the file
ngg - The cursor moves the nth record of the file

G   - The cursor moves the last record of the file
5G   - The cursor moves the 5th record of the file

H   - The cursor moves the starting record of the
       file(window)
L   - The cursor moves the last record of the file
      (window)
M   - The cursor moves the middle record of the
       file(window)

Ctrl + f - To scroll the page forward
Ctrl + b - To scroll the page backward
Ctrl + d - To scroll the half page forward
Ctrl + u - To scroll the half page backward

Esc Mode (Delete Command)

```
x - To delete present cursor position character
X - To delete previous character
dd  - To delete a record
ndd - To delete 'n' records
dw  - To delete a word
5dw - To delete 5 words
db  - To delete a previous word
5db - To delete a 5 previous words
ndb - To delete a n previous words
d$  - To delete present cursor to the end of the record
d^  - To delete present cursor position to starting
      of the record
```

Yanking Commands (copying)

```
y   - To yank a single character
y^  - To yank present cursor position to starting of
      the record
y$  - To yank present cursor position to ending of
      the record
dgg - To yank present cursor position to starting of
      the record
      Ex : (1 – 20) if cursor in 10th position

yy  - To yank the record (copy)
5yy - To yank 5 records present cursor position
nyy - To yank n records present cursor position
ngg - To yank present cursor position to the first position
yG  - To yank present cursor position to the last position
yw  - To yank the work
5yw - To yank 5 words
yb  - To yank previous word
5yb - To yank previous 5 words
nyb - To yank previous n words
```

Paste Commands

```
p   - Paste the right side of the cursor position
P   - Paste the left side of the cursor position

p   - Paste below the cursor position
P   - Paste above the cursor position
```

Replace Commands

```
r   - To replace present cursor position character
R   - To replace the entire record
cc  - To recreate the Record
cw  - To replace the word
cv  - To replace the previous word
```

Special Commands

```
u   - To undo the previous action
.   - To redo the previous action
J   - To join the records
~   - To change to lowercase to upper case letter upper case
      to lowercase letters
```

Search Command

/ <pattren> - To search for particular pattern (word)

```
Ex : /unix

n   - cursor moves to the next accurance
N   - cursor moves to the previous accurance
```

Insert Mode Commands

```
A   - Append the letter end of the line
a   - Append the letter after the cursor
i   - Append the letter before the cursor
I   - Append the letter beginning of the line
o   - open a line below the cursor
O   - open a line above the cursor
s   - Delete the present cursor position and insert
      the character
```

Colon Mode Commands:

```
:w   - To save a file
:q   - To quit a file without save
:wq! - To save and quit a file forcibly
:senu - To display record numbers
:se nonu - To remove display record numbers

:/<pattern> - To search top to bottom
:?<pattern> - To search bottom to top

:se ic   - It ignores the cases and search the records
```

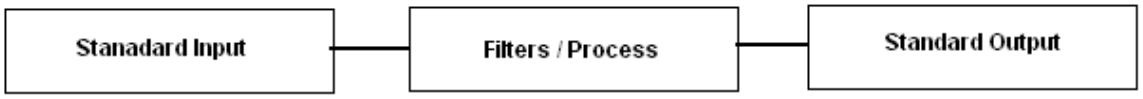
```
:se noic - It disable particular above action
:se ai   - It sets a new line (auto indent)
:se noai - It disables particular action

:d$ - To delete a last record
:%d - To delete all the records in the file

:10,20w <new filename> - Copy to another new file
:10,25w! <target file> - Copy to target file forcibly
:25,30w >> <filename>  - Append existing file
:10 co 50 - Copy 10th record after the 50th Record
:10 mo 50 - Move 10th record to 50th record
:5,15 mo 50 - Move 5, 15 records after 50th record

:s/unix/linux - All records replace
$vi <file1> <file2> <file3> - Multiple file Editor
:n<file3> - to switch file3
:n - instead of file1 to file3
:N - switching from backward file
```

Redirection

```
Ex: $ cat < file1 >file2
It redirects the file1 to file2

Ex: $ cat > file1 <file2
It redirects the file2 to file1

Ex: $ cat >file1 <file2 file3
file2 file3 redirects to file1

Ex: $ cat file1 file2 file3 2>>error
```

FILTERS

Filter is a command or program, which takes the input from the standard input and then process data to give required output



Simple Filters

Advance Filters

```
More                    1. grep
less                    2. sed
pg                      3. awk
head
tail
wc
tee
tr
sort
cut
uniq
```

1. More: More is a filter to display the paging through text page by page

Syntax: $ more <filename>
f - To move next page
b - To move previous page
Spacebar – To scroll the page
Enter Key – To scroll the page line by line
q - Quit

```
Ex: $ cat Imorz | more
```

2. less : Less is a program similar to more (1), but which allows backward movement in the file as well as forward movement. Also, less does not have to read the entire input file before starting, so with large input files it starts up faster than text editors like vi

Syntax: $ less <filename>
f - To move next page
b - To move previous page
Spacebar – To scroll the page
Enter Key – To scroll the page line by line
q – Quit

```
Ex: $ cat file1 | less
```
Note: This command works on Linux only

3. Pg: It displays page by page
Syntax: $ pg <filename>

f - To move next page
b - To move previous page
Spacebar – To scroll the page
Enter Key – To scroll the page line by line

q – Quit

```
Ex: cat venkat | pg
```
Note: This command don't works in Linux

4. head : It displays the first 10 lines of the file (page) – default

Syntax: $ head <filename>

```
Ex: $ head -15 venkat
Ex: $ cat <filename> | head 20 - It display first 20 lines
```

5. tail : It displays last 10 records of the file

Syntax: tail <filename>
: tail -20 <filename>

```
Ex: cat <filename> | tail -20
Ex: tail +25 <filename> - It displays last 25 records
```

In-between records

Records from 15 to 20

```
Ex: $ head -10 <filename> | tail -5 (Records 15 to 20)
Ex: $ tail +30 <filename> | head -5 (Records 30 to 35)
```

If u want to redirect to permanent

```
Ex: $ head -20 <filename> |tail -5 > file1 (file1 is newfile)
Ex: $ tail +30 <filename> |head -5 > file1 (file1 is newfile)
```

6. wc : This filter is used to word count

Syntax: $ wc <filename>
```
Ex: $ wc kiran
```

***no of records to** display*
```
Ex: $ wc -l <filename>
```

no. of words to display
```
Ex: $ wc -w <filename>
```

no. of characters to display

```
Ex: $ wc -c <filename>
```

max – length – characters to disply in a file

```
Ex: $ wc -L <filename>
```

7. tee : It displays and redirects the same time
read from standard input and write to standard output and files

```
Ex: $ ls -l <filename>
    $ ls -l | tee <filename>
    $ ls --full-time | tee -a <filename> - appending
                                          existing file
```

Note: without | (pipe symbol) tee filter don't works

8. tr : Translate characters by characters
Translate or delete characters

Syntax: $ tr [option] <filename>

```
Ex: $ tr "*", "#" <filename

Ex: $ tr " abcd", "xyz" <filename
```

Duplicate characters avoid with sequential order

```
Ex: $ tr -s "r" < filename
```

Delete the characters

```
Ex: $ tr -d "r" < filename
```

9. Sort: sort lines of text files according to the first character

Syntax: sort [option] <filename>
Ex : sort student

student filename

01 vijay 60 90
12 anil 70 75
32 sujan 65 40
04 hari 70 45
22 raju 85 50

Sorting priority

1. Blank space
2. Special character
3. Numeric
4. Uppercase
5. Lowercase

Numeric:

Ex: `$ sort -n <filename>`
`$ Sort -nr <filename> - Numeric and reverse`

If u want permanent
`Ex: $ sort -nr <filename> > new file`

$ sort -t ":" - field separator

According to the second fields

```
$ Sort -dt ":" -k2 - directory sorting

$ ls -l | sort -b – k5 - it takes the blank about Field
                         separator
```

10. Cut: Remove sections from each line of files.

1. Character cutting

```
Ex: $ cut -c <filename>
First character of the all lines

Ex: $ cut -c1 <filename>
To cut the first Character Of the all records

Ex: $ cut -c8 <filename>
To cut the 8th character in each line

Ex: $ cut -c4, 8 <filename>
To cut 4th and 8th Character In each line

Ex: $ cut -c4 -8 <filename>
To cut 4th to 8th Characters range
```

2. Field cutting

-d field separator

```
Ex: $ cut -d ":" -f2 <filename>
Cut 2nd field in all records

Ex: $ cut -d ":" -f2, 5 <filename>
Cut 2nd and 5th records

Ex: $ cut -d ":" -f2, -5 <filename>
Cut 2nd to 5th records

Ex: $ ls -l | tr -s " "
Sequence space

Ex: $ ls -l | tr -s " " | cut -d " " -f5
Sequence and cut
```

11. Uniq: Remove duplicate lines from a sorted file

Syntax: $ uniq [option] <filename>

```
Ex: $ uniq -d <filename >
Displays only the Duplicate records

Ex: $ uniq -D <filename>
Print all duplicate Records and demeliting is done with blank
lines

Ex: $ uniq -c <filename>
Prefix lines by the Number of occurrences
```

Advance Filters

1. Grep

The grep utilities are a family of Unix tools, including grep, egrep, and fgrep, that perform repetitive searching tasks. The tools in the grep family are very similar, and all are used for searching the contents of files for information that matches particular criteria. For most purposes, you'll want to use fgrep, since it's generally the fastest

The general syntax of the grep commands is:

Syntax: grep [-options] pattern [filename]

You can use fgrep to find all the lines of a file that contain a particular word. For example, to list all the lines of a file named my file in the current directory that contain the word "dog", enter at the Unix prompt:

```
Ex: fgrep dog myfile
```

This will also return lines where "dog" is embedded in larger words, such as "dogma" or "dogged". You can use the -w option with the grep command to return only lines where "dog" is included as a separate word:

```
Ex: grep -w dog myfile
```

To search for several words separated by spaces, enclose the whole search string in quotes, for example:

```
Ex:  fgrep "dog named Checkers" myfile
```

The fgrep command is case sensitive; specifying "dog" will not match "Dog" or "DOG". You can use the -i option with the grep command to match both upper- and lowercase letters:

```
Ex:  grep -i dog myfile
```

To list the lines of myfile that do not contain "dog", use the -v option:

```
Ex:  fgrep -v dog myfile
```

If you want to search for lines that contain any of several different words, you can create a second file (named second file in the following example) that contains those words, and then use the -f option:

```
Ex: fgrep -f second file my file
```

You can also use wildcards to instruct fgrep to search any files that match a particular pattern. For example, if you wanted to find lines containing "dog" in any of the files in your directory with names beginning with "my", you could enter:

```
Ex: fgrep dog my*
```

This command would search files with names such as my file, my.hw1, and my stuff in the current directory. Each line returned would be prefaced with the name of the file where the match was found.

By using pipes and/or redirection, you can use the output from any of these commands with other Unix tools, such as more, sort, and cut. For example, to print the fifth word of every

line of my file containing "dog", sort the words alphabetically, and then filter the output through the more command for easy reading, you would enter at the Unix prompt:

```
Ex: fgrep dog myfile | cut -f5 -d" " | sort | more
```

If you want to save the output in a file in the current directory named new file, enter:

```
Ex: fgrep dog my file | cut -f5 -d" " | sort > new file

$ grep –n <file> - To display record number

$ grep -i<file>  - To ignore record

$ grep –c <file> - To count in how many records expression

$ grep –E <file> - To search for multiple expression

$ grep –L <file> - It give the file name and which the
                   regular expression

$ grep –r <file> - To search regressively and present working
                   directory

$ grep –w <file> - To search for the exact match for the word

$ grep –s <file> - To suppress errors

$ grep “jai” <file> - It prints the expressive record which
                      have got the “jai”

$ grep –n “jai” <file> - It display the record number and
                         expression what we give “jai”

$ grep –in “jay” <file> - It ignore and prints the record
                          “Jay”

$ grep –E “jay prem” <file> - It search for multiple
                              expressions and prints the record

$ grep –l “jay”*    - It displays the filename which the
                      “Jay” expression is
```

2. SED

Sed is a stream editor. A stream editor is used to perform basic text transformations on an input stream (a file or input from a pipeline). While in some ways similar to an editor which permits scripted edits (such as ed), sed works by making only one pass over the input(s), and is consequently more efficient. But it is sed's ability to filter text in a pipeline, which particularly distinguishes it from other types of editors.

Sed works as follows: it reads from the standard input, one line at a time. for each line, it executes a series of editing commands, then the line is written to STDOUT. An example that shows how it works: we use the s command. s means "substitute" or search and replace. The format is

```
s/regular-expression/replacement text/{flags}
```

We won't discuss all the flags yet. The one we use below is g, which means, "replace all matches"

```
>cat file
I have three dogs and two cats
>sed -e 's/dog/cat/g' -e 's/cat/elephant/g' file
I have three elephants and two elephants
>
```

OK. So what happened? Firstly, *sed* read in the line of the file and executed

```
s/dog/cat/g
```

Which produced the following text:

```
I have three cats and two cats
```

and then the second command was performed on the *edited line* and the result was

```
I have three elephants and two elephants
```

We actually have a name for the "current text": it is called the *pattern space*. So a precise definition of what *sed* does is as follows:

sed reads the standard input into the pattern space, performs a sequence of editing commands on the pattern space, then writes the pattern space to STDOUT.

Syntax: sed [OPTION]... {Script-only-if-no-other-script} [Input-file]...

OR

Syntax: sed [option] ‘address/action’ <file>

Options:	-n	To suppress input/output
	-e	Multiple Expressions
	-d	Delete
	-p	Print
	-s	Substitute

Firstly, the way you usually use *sed* is as follows:

```
>sed -e 'command1' -e 'command2' -e 'command3' file
>{shell command}|sed -e 'command1' -e 'command2'
>sed -f sedscript.sed file
>{shell command}|sed -f sedscript.sed
```

so *sed* can read from a file or STDIN, and the commands can be specified in a file or on the command line. Note the following:

That if the commands are read from a file, trailing white space can be fatal, in particular, it will cause scripts to fail for no apparent reason. I recommend editing sed scripts with an editor such as vim, which can show end of line characters so that you can "see" trailing white space at the end of line.

Substitute

The format for the substitute command is as follows:

[*address1*[,*address2*]]s*/pattern/replacement/[flags]*

The flags can be any of the following

n replace nth instance of pattern with replacement

p write pattern space to STDOUT if a successful substitution takes place

w file Write the pattern space to file if a successful substitution takes place

Delete

The delete command is very simple in its syntax: it goes like this

```
[address1[ , address2 ] ]d
```

And it deletes the content of the pattern space. All following commands are skipped (after all, there's very little you can do with an empty pattern space), and a new line is read into the pattern space.

Example 1

```
>cat file

http://www.foo.com/mypage.html

>sed -e 's@http://www.foo.com@http://www.bar.net@'
file

http://www.bar.net/mypage.html
```

Note that we used a different delimiter, @ for the substitution command. Sed permits several delimiters for the s command including @%,;: these alternative delimiters are good for substitutions which include strings such as filenames, as it makes your sed code much more readable.

Example 2

```
>cat file

the black cat was chased by the brown dog

>sed -e 's/black/white/g' file

the white cat was chased by the brown dog
```

That was pretty straightforward. Now we move on to something more interesting.

Example 3

```
>cat file

the black cat was chased by the brown dog.
the black cat was not chased by the brown dog

>sed -e '/not/s/black/white/g' file

the black cat was chased by the brown dog.
the white cat was not chased by the brown dog.
```

In this instance, the substitution is only applied to lines matching the regular expression `not`. Hence it is not applied to the first line.

Example 4

```
>cat file

line 1 (one)
line 2 (two)
line 3 (three)
```

Example 4a

```
>sed -e '1,2d' file
line 3 (three)
```

Example 4b

```
>sed -e '3d' file
line 1 (one)
line 2 (two)
```

Example 4c

```
>sed -e '1,2s/line/LINE/' file

LINE 1 (one)
LINE 2 (two)
line 3 (three)
```

Example 4d

```
>sed -e '/^line.*one/s/line/LINE/' -e '/line/d'
file

LINE 1 (one)
```

3a : This was pretty simple: we just deleted lines 1 to 2.
3b : This was also pretty simple. We deleted line 3.
3c : In this example, we performed a substitution on lines 1-2.
3d : now this is more interesting, and deserves some explanation. Firstly, it is clear that line 2 and 3 get deleted. But let's look closely at what happens to line 1.
First, line 1 is read into the pattern space. It matches the regular expression `^line.*one`
So the substitution is carried out, and the resulting pattern space looks like this:

```
LINE 1 (one)
```

So now the second command is executed, but since the pattern space does not match the regular expression `line`, the delete command is not executed.

Example 5

```
>cat file

hello
this text is wiped out
Wiped out
hello (also wiped out)
WiPed out TOO!
goodbye
(1) This text is not deleted
(2) neither is this ... ( goodbye )
(3) neither is this
hello
but this is
and so is this
and unless we find another g**dbye
every line to the end of the file gets deleted

>sed -e '/hello/,/goodbye/d' file


(1) This text is not deleted
(2) neither is this ... (goodbye)
(3) neither is this
```

This illustrates how the addressing works when two pattern addresses are specified. sed finds the first match of the expression "hello", deleting every line read into the pattern space until it gets to the first line after the expression "goodbye". It doesn't apply the delete command to any more addresses until it comes across the expression "hello" again. Since the expression "goodbye" is not on any subsequent line, the delete command is applied to all remaining lines.

AWK

AWK is a simple and elegant pattern scanning and processing language
AWK is also the most portable scripting language

It was created in late 70th of the last century. The name was composed from the initial letters of three original authors Alfred V. Aho, Brian W. Kernighan, and Peter J. Weinberger. It is commonly used as a command-line filter in pipes to reformat the output of other commands. It's the precursor and the main inspiration of Perl. Although originated in Unix it is available and widely used in Windows environment too.

AWK takes two inputs: data file and command file. The command file can be absent and necessary commands can be passed as augments. As Ronald P. Loui aptly noted ***awk is very under appreciated language:***

The main advantage of AWK is that unlike Perl and other "scripting monsters" that it is very slim without feature creep so characteristic of Perl and thus it can be very efficiently used with pipes. Also it has rather simple, clean syntax and like much heavier TCL can be used with C for "dual-language" implementations.

awk's favor compared to perl:

- awk is simpler (especially important if deciding which to learn first)
- awk syntax is far more regular (another advantage for the beginner, even without considering syntax-highlighting editors)
- you may already know awk well enough for the task at hand
- you may have only awk installed
- awk can be smaller, thus much quicker to execute for small programs
- awk variables don't have `$' in front of them :-)
- clear perl code is better than unclear awk code; but NOTHING comes close to unclear perl code

The basic function of awk is to search files for lines (or other units of text) that contain certain patterns. When a line matches one of the patterns, awk performs specified actions on that line. awk keeps processing input lines in this way until it reaches the end of the input files

Syntax : awk [option] 'selection criteria {action}' <file>

Options : -F - To specify the field separator
-f - To invoke the source code
{action} - Only the print action

predefined variables in awk

all predefined variables are in upper cases

```
FS      - Input field separator
OFS     - Ouput field separator
NF      - Number of fields
NR      - Record numbers or No. of records
$       - Fields in awk
```

Comparation Operator in awk

```
>       - Grater than
```

```
>=     - Grater than equal
<      - Less than
<=     - Less than equal
==     - Equal to
!=     - Not equal
~      - Matching
!~     - not matching
```

Logical Operator

&& - AND
|| - OR

awk has got 3 sections

1. BIGIN
2. MIDDLE
3. END

Begin is keyword for the begin section the variable can be assign in begin section

All the operator in the middle sections, Middle is not keyword for the middle section

What ever u print every thing in the section, End is the keyword for the end section

```
$ awk  '/ajay/{print}'<file>
It prints the all the records

$ awk '/ajay|ramu/{print}'<file>
To search for multiple expressions and print

$ awk 'NR==4{print}' <file>
To print specific record

$ awk 'NR==3,NR==7{print}'<file>
To print range of records

$ awk 'NR>4{print}' <file>
To print all the records which are >4

$ awk 'NR>={print}' <file>

$ awk 'NR<4{print}' <file>
To print all the records which are <4

$ awk 'NR<=4{print}' <file>
```

if u want print only specific fields

```
$ awk –F ":" 'NR==4 {print $1, $3, $4}' <file>
```

simple `awk` program emulates the `cat` utility; it copies whatever you type on the keyboard to its standard output (why this works is explained shortly).

```
$ awk '{ print }'
Now is the time for all good men
-| Now is the time for all good men
to come to the aid of their country.
-| to come to the aid of their country.
Four score and seven years ago, ...
-| Four score and seven years ago, ...
What, me worry?
-| What, me worry?
Ctrl-d
```

- Print the length of the longest input line:

```
awk '{if(length($0)> max)max = length($0) }
                END { print max }' data
```

- Print every line that is longer than 80 characters:

```
awk 'length($0) > 80' data
```

The sole rule has a relational expression as its pattern and it has no action—so the default action, printing the record, is used.

- Print the length of the longest line in data:

```
expand data | awk '{if x < length()) x =length()}
   END {print "maximum line length is " x}'
```

The input is processed by the expand utility to change tabs into spaces, so the widths compared are actually the right-margin columns.

- Print every line that has at least one field:

```
awk 'NF > 0' data
```

This is an easy way to delete blank lines from a file (or rather, to create a new file similar to the old file but from which the blank lines have been removed).

- Print seven random numbers from 0 to 100, inclusive:

```
awk 'BEGIN {for(i=1;i<=7;i++)print int(101*rand())}'
```

- Print the total number of bytes used by *files*:

```
ls -l files | awk '{ x += $5 }
      END { print "total bytes: " x }'
```

- Print the total number of kilobytes used by *files*:

```
ls -l files | awk '{ x += $5 }
   END { print "total K-bytes: " x + 1023)/1024 }'
```

- Print a sorted list of the login names of all users:

```
awk -F: '{ print $1 }' /etc/passwd | sort
```

- Count the lines in a file:

```
awk 'END { print NR }' data
```

- Print the even-numbered lines in the data file:

```
awk 'NR % 2 == 0' data
```

If you use the expression `NR % 2 == 1' instead, the program would print the odd-numbered lines

EXAMPLES

```
# is the comment character for awk.  'field' means
'column'

# Print first two fields in opposite order:
  awk '{ print $2, $1 }' file


# Print lines longer than 72 characters:
```

```
  awk 'length > 72' file


# Print length of string in 2nd column
  awk '{print length($2)}' file


# Add up first column, print sum and average:

       { s += $1 }
 END   { print "sum is", s, " average is", s/NR }


# Print fields in reverse order:

awk '{for i = NF; i > 0; --i)print $i }' file


# Print the last line
      {line = $0}
  END {print line}

# Print the total number of lines that
  contain the word Pat

  /Pat/ {nlines = nlines + 1}
  END {print nlines}

# Print all lines between start/stop pairs:
  awk '/start/, /stop/' file


# Print all lines whose first field is
  different from previous one:

  awk '$1 != prev { print; prev = $1 }' file


# Print column 3 if column 1 > column 2:
  awk '$1 > $2 {print $3}' file


# Print line if column 3 > column 2:
  awk '$3 > $2' file
```

```
# Count number of lines where col3 >col 1
  awk '$3 > $1 {print i + "1"; i++}' file


# Print sequence number and then column 1 of file:
  awk '{print NR, $1}' file


# Print every line after erasing the 2nd field
  awk '{$2 = ""; print}' file


# Print hi 28 times
  yes | head -28 | awk '{ print "hi" }'


# Print hi.0010 to hi.0099 (NOTE IRAF USERS!)
  yes | head -90 | awk '{printf("hi00%2.0f \n",
                                      NR+9)}'


# Replace every field by its absolute value
  { for (i = 1; i <= NF; i=i+1) if ($i < 0)
                          $i = -$i print}

# If you have another character that delimits
fields, use the -F option
# For example, to print out the phone number for
Jones in the following file,
# 000902|Beavis|Theodore|333-242-2222|149092
# 000901|Jones|Bill|532-382-0342|234023
# ...
# type
  awk -F"|" '$2=="Jones"{print $4}' filename


# Some looping for printouts
  BEGIN{
       for (i=875;i>833;i--){
             printf "lprm -Plw %d\n", i
       } exit
        }

Formatted printouts are of the form printf(
"format\n", value1, value2, ... valueN)
```

```
            e.g. printf("howdy %-8s What it is bro.
%.2f\n", $1, $2*$3)
      %s = string
      %-8s = 8 character string left justified
      %.2f = number with 2 places after .
      %6.2f = field 6 chars with 2 chars after .
      \n is newline
      \t is a tab

# Print frequency histogram of column of numbers
$2 <= 0.1 {na=na+1}
($2 > 0.1) && ($2 <= 0.2) {nb = nb+1}
($2 > 0.2) && ($2 <= 0.3) {nc = nc+1}
($2 > 0.3) && ($2 <= 0.4) {nd = nd+1}
($2 > 0.4) && ($2 <= 0.5) {ne = ne+1}
($2 > 0.5) && ($2 <= 0.6) {nf = nf+1}
($2 > 0.6) && ($2 <= 0.7) {ng = ng+1}
($2 > 0.7) && ($2 <= 0.8) {nh = nh+1}
($2 > 0.8) && ($2 <= 0.9) {ni = ni+1}
($2 > 0.9) {nj = nj+1}
END {print na, nb, nc, nd, ne, nf, ng, nh, ni, nj,
NR}

# Find maximum and minimum values present in
  column 1
NR == 1 {m=$1 ; p=$1}
$1 >= m {m = $1}
$1 <= p {p = $1}
END { print "Max = " m, "   Min = " p }

# Example of defining variables, multiple
  commands on one line

NR == 1 {prev=$4; preva = $1; prevb = $2; n=0;
sum=0}
$4 != prev {print preva, prevb, prev, sum/n; n=0;
sum=0; prev = $4; preva = $1; prevb = $2}
$4 == prev {n++; sum=sum+$5/$6}
END {print preva, prevb, prev, sum/n}

# Example of using substrings
# substr($2,9,7) picks out characters 9 thru 15 of
column 2
{print "imarith", substr($2,1,7) " - " $3,
"out."substr($2,5,3)}
```

```
{print "imarith", substr($2,9,7) " - " $3,
"out."substr($2,13,3)}
{print "imarith", substr($2,17,7) " - " $3,
"out."substr($2,21,3)}
 print "imarith", substr($2,25,7) " - " $3,
"out."substr($2,29,3)}
```

1. Renaming within the name:

```
ls -1 *old* | awk '{print "mv "$1" "$1}' | sed
s/old/new/2 | sh
```

(although in some cases it will fail, as in file_old_and_old)

2. Remove only files:

```
ls -l * | grep -v drwx | awk '{print "rm "$9}' | sh
```

or with awk alone:

```
ls -l|awk '$1!~/^drwx/{print $9}'|xargs rm
```

Be careful when trying this out in your home directory. We remove files!

3. Remove only directories

```
ls -l | grep '^d' | awk '{print "rm -r "$9}' | sh
```

or

```
ls -p | grep /$ | wk '{print "rm -r "$1}'
```

or with awk alone:

```
ls -l|awk '$1~/^d.*x/{print $9}'|xargs rm -r
```

Be careful when trying this out in your home directory. We remove things!

4. Killing processes by name (in this example we kill the process called netscape):

```
kill `ps auxww | grep netscape | egrep -v grep | awk
'{print $2}'`
```

Environment Control

Command	Description
cd d	Change to directory d
mkdir d	Create new directory d
rmdir d	Remove directory d
mv f1 [f2...] d	Move file f to directory d
mv d1 d2	Rename directory d1 as d2
passwd	Change password
alias name1 name2	Create command alias (csh/tcsh)
alias name1="name2"	Create command alias (ksh/bash)
unalias name1[na2...]	Remove command alias na
ssh nd	Login securely to remote node
exit	End terminal session
setenv name v	Set env var to value v (csh/tcsh)
export name="v"	Set environment variable to value v(ksh/bash)

Output, Communication, & Help

Command	Description
lpr -P printer f *or* lp -d printer f	Output file f to line printer
script [f]	Save terminal session to f
exit	Stop saving terminal session
mailx username	Send mail to user
man name	Unix manual entry for name

Process Control

Command	Description
CTRL/c *	Interrupt processes
CTRL/s *	Stop screen scrolling
CTRL/q *	Resume screen output
sleep n	Sleep for n seconds
jobs	Print list of jobs
kill %	Kill job n
ps	Print process status stats
kill -9 n	Remove process n
CTRL/z *	Suspend current process
stop %n	Suspend background job n
cmmd&	Run cmmd in background
bg [%n]	Resume background job n
fg [%n]	Resume foreground job n
exit	Exit from shell

Environment Status

Command	Description
ls [d] [f...]	List files in directory
ls -1 [f...]	List files in detail
alias [name]	Display command aliases
printenv [name]	Print environment values
quota	Display disk quota
date	Print date & time

who	List logged in users
whoami	Display current user
finger [username]	Output user information
chfn	Change finger information
pwd	Print working directory
history	Display recent commands
! n	Submit recent command n

File Manipulation

Command	Description
vi [f]	Vi full screen editor
emacs [f]	Emacs full screen editor
ed [f]	Text editor
wc f	Line, word, & char count
cat f	List contents of file
more f	List file contents by screen
cat f1 f2 >f3	Concatenates f1 & f2 into f3
chmod mode f	Change protection mode of f
cmp f1 f2	Compare two files
cp f1 f2	Copy file f1 into f2
sort f	Alphabetically sort f
split [-n] f	Split f into n-line pieces
mv f1 f2	Rename file f1 as f2
rm f	Delete (remove) file f

grep 'ptn' f	Outputs lines that match ptn
diff f1 f2	Lists file differences
head f	Output beginning of f
tail f	Output end of f

Compiler

Command	Description
cc [-o f1] f2	C compiler
lint f	Check C code for errors
f77 [-o f1] f2	Fortran77 compiler
pc [-o f1] f2	Pascal compiler

Abbreviations used in this document

CTRL/x	hold down control key and press x
d	directory
env	environment
f	filename
n	number
nd	computer node
prtr	printer
ptn	pattern
var	variable
[y/n]	yes or no