‘global search for the regular expression’: The grep command is a filter that is used to search for lines  
matching a specified pattern and print the matching lines to standard output.

1) Match all lines that start with‘hello’.E.g:“hellothere” ->   
    grep “^hello” file1  
2) Match all lines that end with‘done’.E.g:“welldone” ->   
    grep “done$” file1  
3) Match all lines that contain any of the letters‘a’,‘b’,‘c’,‘d’or‘e’ ->   
    grep “[a-e]” file1  
4) Match all lines that do not contain a vowel ->   
    grep “[^aeiou]” file1  
5) Match all lines that start with a digit following zero or more spaces. E.g:“1.”or“2.” ->   
    grep “ \*[0-9]” file1  
6) Searching in all files recursively using ->   
    grep -r "ramesh" \*  
7) Search multiple patterns ->   
    grep -v -e "pattern" -e "pattern"

**find**  
    find command can be used to find files and directories and perform subsequent operations on them. It supports searching by file, folder, name, creation date, modification date, owner and permissions. By using the ‘-exec’ other commands can be executed on files or folders found.  
      
SYNTAX: find <location\_to\_find> [options]

1) Searcha file with specific name ->   
    find . -name file.txt  
3) Searcha file with specific name with ignorecase ->   
    find . -iname file.txt  
2) searcha files in multiple directories ->   
    find . /home /user -name file.txt  
4) Search only files containing name ->   
    find . -type f -iname file.txt  
5) Search only directories containing name ->   
    find . -type d -iname file.txt  
6) Search for empty files and directories ->   
    find . -empty  
7) Search for file with permissions(655) ->   
    find . -perm 655  
8) Search text within multiple files.   
    find./-typef-name"\*.txt"-execgrep'search\_string' {}\;  
9) find files by last modification time ->   
    find . -mtime days  
    # 24 hours -> find . -mtime 1  
    # modified less than 7 days (7 days to till) -> find . -mtime -7  
    # Last 50-100 Days Modified Files -> find . -mtime +50 –mtime -100   
10) Find Last 50 Days Accessed Files -> find . -atime 50  
11) Find Changed Files in Last 1 Hour -> find / -mmin -60  
12) Find Accessed Files in Last 1 Hour -> find / -amin -60

**sed**  
SED command in UNIX is stands for stream editor and it can perform lots of function on file like, searching, find and replace, insertion or deletion.

Consider the following text file as the input file for all cases below. (file.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) Replacing or substituting string   
    sed 's/unix/linux/' file.txt  
2) Replacing the nth occurrence of a pattern in a line   
    sed 's/unix/linux/2' file.txt  
3) Replacing all the occurrence of the pattern in a line   
    sed 's/unix/linux/g' file.txt  
4) Replacing from nth occurrence to all occurrences in a line   
    sed 's/unix/linux/3g' file.txt  
5) Replacing string on a specific line number   
    sed '3 s/unix/linux/' file.txt  
6) Printing only the replaced lines   
    sed -n 's/unix/linux/p' file.txt  
7) Deleting lines from a particular file  
    To Delete a particular line, say 5 in this example  
        sed '5d' file.txt  
    To Delete a last line  
        sed '$d' file.txt  
    To Delete line from range x to y  
        sed '3,6d' file.txt  
    To Delete from nth to last line  
        sed '12,$d' file.txt  
    To Delete pattern matching line  
        sed '/pattern/d' file.txt  
8) Print specific line from a file   
    sed -n '2p' file.txt

**CUT**  
The cut command in UNIX is a command for cutting out the sections column with using delimiter from each line of files and writing the result to standard output.  
    cut -d "delimiter" -f (field number) file.txt  
1) To cut the data with “ “ (space) as delimiter and print first column data  
    cut -d " " -f 1 state.txt  
2) To cut the data with “ “ (space) as delimiter and print first to fourth column data  
    cut -d " " -f 1-4 state.txt

**awk**

The awk command in UNIX is a command for cutting out the sections column with using field separator from each line of files and writing the result to standard output.

Consider the following text file as the input file for all cases below.  
$cat > employee.txt  
ajay manager account 45000   
sunil clerk account 25000   
varun manager sales 50000   
amit manager account 47000

1) Default behavior of Awk: By default Awk prints every line of data from the space separated field of file.  
    awk '{print $1}' employee.txt  
            (or)  
    awk -f “ “ ‘{print $1}’ employee.txt

2) To print the last column of a file using NF – Number of Fields/Columns  
    How to get the last word from a line in file.  
        awk  ‘{print $NF}’ employee.txt

**seq 4**

By default, seq command starts printing the sequence of numbers from 1, if not specified otherwise.

**pipe (|)**

Pipes help combine two or more commands and are used to feed the output of one command as input to another command.

cat file.txt | grep ‘<pattern>’

find . -type f -iname file\* | xargs -I{} rm {}

**date**  
- date command is used to display the system date and time.  
- date command is also used to set date and time of the system.   
- By default the date command displays the date in the time zone on which unix/linux operating system is configured.   
- You must be the super-user (root) to change the date and time.

a) Date and time of 2 years ago.    date --date="2 year ago"    Wed May 26 12:01:31 UTC 2021  
b) Date and time of 5 seconds ago.    date --date="5 sec ago"        Fri May 26 12:01:37 UTC 2023  
c) Date and time of previous day.    date --date="yesterday"        Thu May 25 12:01:51 UTC 2023  
d) Date and time of 2 months ago.    date --date="2 month ago"    Sun Mar 26 12:02:00 UTC 2023  
e) Date and time of 10 days ago.    date --date="10 day ago"    Tue May 16 12:02:10 UTC 2023  
a) Upcoming particular week day.    date --date="next tue"        Tue May 30 00:00:00 UTC 2023  
b) Date and time after two days.    date --date="2 day"        Sun May 28 12:03:03 UTC 2023  
c) Date and time of next day.        date --date="tomorrow"        Sat May 27 12:03:11 UTC 2023  
d) After 1 year on the current day.    date --date="1 year"        Sun May 26 12:03:20 UTC 2024

Formating   
date "+%D"        05/26/23  
date "+%D %T"        05/26/23 05:49:30  
date "+%Y-%m-%d"    2023-05-26  
date "+%Y/%m/%d"    2023/05/26  
date "+%A %B %d %T %y"    Friday May 26 05:49:30 23

Time

time command displays how long it takes to execute a command. It helps in checking the performance of the scripts and commands.

**To change the Timezone in Linux**

**Method1: sudo timedatectl set-timezone Asia/Calcutta**

                    1. timedatectl list-timezones

                    2. sudo timedatectl set-timezone <timezone\_required\_from\_prvious\_command>

**Method2:**

                    1. sudo rm -rf /etc/localtime  
                    2. sudo ln -s /usr/share/zoneinfo/Asia/Kolkata /etc/localtime

**Method3: export TZ=Asia/Calcutta**

                    1. export TZ=<timezone\_required>

basename    
- Returns trailing name component of path  
  basename /user/ubuntu/linux/file.txt   
      output: file.txt  
dirname    
- Returns prefixed path of the component  
  dirname /user/ubuntu/linux/file.txt   
      output: /user/ubuntu/linux

free  
Free command used to check the used and available space of physical memory and swap memory (ram memory) in KB. The free command displays:  
•    Total amount of free and used physical memory  
•    Total amount of swap memory in the system  
•    Buffers and caches used by the kernel

top  
The top command also gives you a real-time update on how much of your swap space is being used.

/proc/meminfo  
    cat /proc/meminfo will contain dynamic information about the kernel and the system.  
      
du  
“du” (Disk Usage) is a standard Unix/Linux command, used to check the information of disk usage of files and directories on a machine.  
1) Using “-h” option with “du” command provides results in “Human Readable Format“. Means you can see sizes in Bytes, Kilobytes, Megabytes, Gigabytes etc.  
        du -h

df  
The command df stands for "disk filesystem". With the -h option (df -h) it shows the disk space in "human readable" form

ps  
ps command is used to display information about the processes that are running in the system.  
1) To view current running processes.  
    ps -ef      
kill  
Use kill command to terminate a process. First get the process id using ps -ef command, then use kill -9 to kill the running Linux process as shown below.   
STEP1: Before kill first get the pid (process id to kill)  
    ps -ef | grep process\_name  
      
    OUTPUT:  
        user 7243 7222 9 22:43 pts/2 00:00:00 process

STEP2: To kill the above process (use the pid from the second column)  
    kill -9 7243

**Crontab (Job Scheduler)**  
The crontab is a list of commands that you want to run on a regular schedule, and also the name of the command used to manage that list. Crontab stands for “cron table, ” because it uses the job scheduler cron to execute tasks; cron itself is named after “chronos, ” the Greek word for time.cron is the system process which will automatically perform tasks for you according to a set schedule. The schedule is called the crontab, which is also the name of the program used to edit that schedule.

Linux Crontab Format:    \*    \*    \*    \*    \*    command to execute

               MIN    HOUR     DOM     MON     DOW           CMD

FIELD    DESCRIPTION    ALLOWED VALUE  
MIN    Minute field    0 to 59  
HOUR    Hour field    0 to 23  
DOM    Day of Month    1-31  
MON    Month field    1-12  
DOW    Day Of Week    0-6  
CMD    Command    Any command to be executed.

To view the Crontab entries: crontab -l  
To edit Crontab Entries: crontab -e  
To edit crontab entries of other Linux users: crontab -u username -e

example  
1) To schedule a job for every minute using Cron  
    \* \* \* \* \* command/script  
2) How to Execute a Linux Cron Jobs Every Second Using Crontab.  
    You cannot schedule an every-second cronjob. Because in cron the minimum unit you can specify is minute.   
3) To schedule a background Cron job for every 10 minutes.  
    \*/10 \* \* \* \* /home/maverick/check-disk-space  
4) Schedule a Job for More Than One Instance (e.g. Twice a Day)  
    executes the specified script at 11:00 and 16:00 on every day  
    00 11,16 \* \* \* /home/ramesh/bin/incremental-backup  
5) Schedule a Job for Specific Range of Time (e.g. Only on Weekdays)  
    This example checks the status of the database everyday (including weekends) during the working hours 9 a.m – 6 p.m  
    \* \* \* \* 1-5 /home/ramesh/bin/check-db-status  
6) Cron job to run on the last day of the month  
       \* \* \* \* \* [[ "$(date --date=tomorrow +\%d)" == "01" ]] && myscript.sh

         \* \* \* \* \* [[  $(date -d +1day +%d) -eq  1  ]] && myscript.sh  
  
7) Cron special keywords and its meaning  
    Keyword    Equivalent  
    @yearly    0 0 1 1 \*  
    @daily    0 0 \* \* \*  
    @hourly    0 \* \* \* \*  
    @reboot    Run at startup.  
      
    a) To schedule a job for first minute of every year using @yearly  
        @yearly /home/maverick/bin/annual-maintenance  
    b) To schedule a Cron job beginning of every month using @monthly  
        @monthly /home/maverick/bin/tape-backup  
    c) To schedule a background job every day using @daily  
        @daily /home/maverick/bin/cleanup-logs "day started"  
    d) To execute a linux command after every reboot using @reboot  
        @reboot CMD

**Shebang**

**A shebang (#!)** is a special line at the beginning of a script that tells the operating system which interpreter to use when executing the script. (for BASH   - #!/bin/bash)

**Variables**

* A bash variable is a temporary storage location for a value.
* Variables can store any data, including strings, numbers, and lists.   
  1) Setting a value for a variable.  
      variable\_name=value  
        
  2) Reading the value for a variable.  
      For referring to or reading a variable we place a $ sign before the variable name.  
          echo "$variable\_name"

**Single quote ('') vs Double quote ("")**

var=10

     ('')   
        - Considers all inside single quotes as String   
        - The $var variable output does not get replaced with the variable value, but displays the variable name within the quotes, i.e., its literal form.

        >$  echo '$var'  
        output:    $var

    ("")   
        - Looks for assigned values to variable and executes the content.   
        - Running the echo command with double quotes expands the $var variable and outputs the assigned value instead of printing the characters within the quotes.

        >$  echo "$var"  
        output:    10

**Special Variables**

**$0 -** The filename of the current script.  
**$# -**The number of arguments supplied to a script.  
**$\* -**All the arguments passed to the script where arguments are space separated and passed as a single string.  
**$@ -**All the arguments passed to the script and it preserves the individual arguments as separate entity.  
**$? -** The status of the last command executed (zero for success and non-zero for failure).  
**$$ -** PID of the current shell. In Bash script, this is the process ID under which they are executing.  
**$! -**PID of the process which is recently sent to background.

**Escape character**

**\**  
        - A non-quoted backslash ‘\’ is the Bash escape character.   
        - It preserves the literal value (considers only as string) of the following next character and makes it a string only.

        Below only the character $ will be escaped and treated as string   
            >$  echo "\$var"  
            output:    $var

        - If a \n pair appears, and the backslash itself is not quoted, a newline will be added.

            echo -e "\nThis is to add newline before this echo"

            Multiple newlines   
            echo -e "\nThis is \n to add newline \n before this echo\n"

**Write to a File using the Redirection Operator (>)**

* Redirection allows you to capture the output from a command and overwrite it as input to another command or file.
* It will overwrite/Erase all the content of file and add the new content to file.
* What ever the content it will written to file and nothing will be print in console.

        echo "This is Redirection of text to file" > file.txt

        To write to a file and also to print the written content in console/terminal   
             echo "This is Redirection of text to file" | Tee file.txt

**Append to a File using the double Redirection Operator (>>)**

* Double Redirection allows you to capture the output from a command and send it as input to another command or file.
* It will append the content to the end of file by keeping the old content.

        echo "This is Redirection of text to file" >> file.txt

        To append to a file and also to print the written content in console/terminal   
             echo "This is Redirection of text to file" | Tee -a file.txt  
                                         (OR)  
             echo "This is Redirection of text to file" | Tee --append file.txt