**Who:** Whenever a user needs to know about how many users are using or are logged-in into a particular Linux-based operating system, he/she can use the "who" command to get that information.

*Output: logged-in users' names, last reboot time*

-a: display all details of currently logged in users

-b: last system boot

**whoami :** Print the effective username of the user who ran whoami. {Alternate:w, who, pinky}

**date:** display time

option: -u, $date --date="2 year ago" or "2 sec ago" or ‘’yesterday’’

**cal:** cal -y, cal 08 2020, cal 2018, cal -3

**mkdir:** To create Directory/folder

**rmdir :** To remove Directory/folder

**cd :** single dot current directory, double dot parent directory

**pwd:** print the present working directory

**cp :** cp a.txt b.txt, cp a.txt b.txt new, cp -R Src\_directory Dest\_directory, cp -i a.txt b.txt, cp \*.txt Folder1, similarly for folders

**rm:** rm a.txt …, rm -f e.txt, rm -r \*

**mv:** (i) It renames a file or folder. (ii) It moves a group of files to a different directory.

mv [Option] source destination

mv a.txt b.txt(renamed), option:-i, -f

**ls :** list

ls -t : It sorts the file by modification time, showing the last edited file first. head -1 picks up this first file.To open the last edited file in the current directory use the combination of ls and head commands as shown below.

Display One File Per Line Using ls -1

Display All Information About Files/Directories Using ls -l

* - normal file
* d : directory
* s : socket file
* l : link file

1. Field 1 – File Permissions: Next 9 character specifies the files permission. The every 3 characters specifies read, write, execute permissions for user(root), group and others respectively in order. Taking above example, -rw-rw-r– indicates read-write permission for user(root) , read permission for group, and no permission for others respectively. If all three permissions are given to user(root), group and others, the format looks like -rwxrwxrwx
2. Field 2 – Number of links: Second field specifies the number of links for that file. In this example, 1 indicates only one link to this file.
3. Field 3 – Owner: Third field specifies owner of the file. In this example, this file is owned by username ‘maverick’.
4. Field 4 – Group: Fourth field specifies the group of the file. In this example, this file belongs to ”maverick’ group.
5. Field 5 – Size: Fifth field specifies the size of file in bytes. In this example, ‘1176’ indicates the file size in bytes.
6. Field 6 – Last modified date and time: Sixth field specifies the date and time of the last modification of the file. In this example, ‘Feb 16 00:19’ specifies the last modification time of the file.
7. Field 7 – File name: The last field is the name of the file. In this example, the file name is 1.c.

Display Order Files Based on Last Modified Time Using ls -lt

Display File Size in Human Readable Format Using ls -lh

Display Hidden Files Using ls -a

**touch** : Used to create file

* touch file\_name, touch File1\_name File2\_name File3\_name, touch -a filename(to change access time)
* touch -c : This command is used to check whether a file is created or not. If not created then don’t create it. This command avoids creating files.
* touch -t YYMMDDHHMM fileName

**cat:** Cat(concatenate) command is very frequently used in Linux. It reads data from the file and gives their content as output. It helps us to create, view, concatenate files. So let us see some frequently used cat commands.

* $cat filename, view single file, $cat file1 file2, $cat -n filename(with line no)
* $cat > filename
* $cat [filename-whose-contents-is-to-be-copied] > [destination-filename]
* $cat file1 >> file2 (append)
* $tac filename in reverse order
* $cat \*.txt view all the files in the folder
* $cat >> geeks.txt, write already existing file.

**chmod :** In Unix-like operating systems, the chmod command is used to change the access mode of a file. The name is an abbreviation of change mode.

To use chmod to set permissions, we need to tell it:

* Who: Who we are setting permissions for.
* What: What change are we making? Are we adding or removing the permission?
* Which: Which of the permissions are we setting?

We use indicators to represent these values, and form short “permissions statements” such as u+x, where “u” means ” user” (who), “+” means add (what), and “x” means the execute permission (which).

The “who” values we can use are:

* u: User, meaning the owner of the file.
* g: Group, meaning members of the group the file belongs to.
* o: Others, meaning people not governed by the u and g permissions.
* a: All, meaning all of the above.

If none of these are used, chmod behaves as if “a” had been used.

The “what” values we can use are:

–: Minus sign. Removes the permission.

+: Plus sign. Grants the permission. The permission is added to the existing permissions. If you want to have this permission and only this permission set, use the = option, described below.

=: Equals sign. Set a permission and remove others.

The “which ” values we can use are:

r: The read permission.

w: The write permission.

x: The execute permission.

chmod u=rw,og=r new\_file.txt

chmod a+x new\_script.sh can add the execute permission for everyone

The digits you can use and what they represent are listed here:

0: (000) No permission.

1: (001) Execute permission.

2: (010) Write permission.

3: (011) Write and execute permissions.

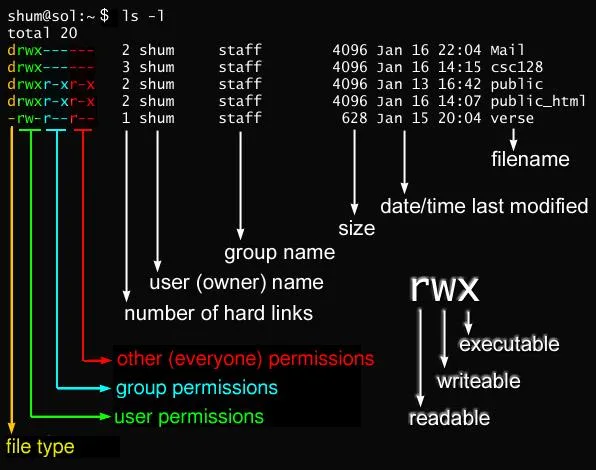
4: (100) Read permission.

5: (101) Read and execute permissions.

6: (110) Read and write permissions.

7: (111) Read, write, and execute permissions.

chmod 664 \*.page



**stat file/folder:** Information regarding file or folder

**umask :** umask command returns, or sets, the value of the system's file mode creation mask

umask : returns the current status of system file creation

|  |  |  |
| --- | --- | --- |
| **umask digit** | **default file permissions** | **default directory permissions** |
| 0(000) | rw- | rwx |
| 1(001) | rw- | rw- |
| 2(010) | r-- | r-x |
| 3(011) | r-- | r-- |
| 4(100) | -w- | -wx |
| 5(101) | -w- | -w- |
| 6(110) | --- | --x |
| 7(111) | no permission | no permission |

To view your system's current **umask** value, enter the command:

umask

which returns your system's umask as a four-digit octal number, for example:

0002

first digit:

SGID means that new files inside this dir will inherit group owner.

SUID Mostly ignored in Linux and Unix. BSD varies.

Sticky Protect files inside from being modified by a different user.

Again, the first zero is a special permission digit and can be ignored; for our purposes, **0002** is the same as **002**.

umask -S : view this as a symbolic representation, use the **-S** flag, Which returns the same value symbolically, for example:u=rwx,g=rwx,o=rx

**umask 777 : new files inaccessible to everyone - no one can read, write, or execute them.**

**—------------------------------------------------------------------------------------------------------------------**

**bc :** bc command is used for command line calculator. It is similar to basic calculator by using which we can do basic mathematical calculations.

Input : echo 12+5| bc, or you can directly run bc command to open calculator.

Output : 17

Input : echo "10^2" | bc

Output : 100

Input:x=`echo "12+5" | bc`

echo $x

Input: echo "var=10;var" | bc

Output: 10

Input: echo "var=10;var^=2;var" | bc

Output: 100

**expr :** The expr command in Unix evaluates a given expression and displays its corresponding output

expr 12 + 8

expr 12 \\* 2

we can use this in shell script: ex addition

echo "Enter two numbers"

read x

read y

sum=`expr $x + $y`

echo "Sum = $sum"

single comma act as command substitution

**factor :** used to print the prime factors of the given numbers, either given from command line or read from standard input.

factor 100

**logname:** Print the logged username , extension of this w, who whoami

**uname :** most commonly used to determine the processor architecture, the system hostname and the version of the kernel running on the system.

options: -s, -r, -v, -m, -, -i, -o

**tty :** prints the file name of the terminal connected to standard input.(teletype)

In laymen terms the primary difference between TTY and PTS is the type of connection to the computer. TTY ports are direct connections to the computer such as a keyboard/mouse or a serial connection to the device. PTS connections are SSH connections or telnet connections. All of these connections can connect to a shell which will allow you to issue commands to the computer.

**df:** The df command (short for disk free), is used to display information related to file systems about total space and available space.

**-a, –all :** includes pseudo, duplicate and inaccessible file systems.

**-B, –block-size=SIZE :** scales sizes by SIZE before printing them.

**-h, –human-readable :** print sizes in power of 1024

**-H, –si:** print sizes in power of 1000

**-i, –inodes :** list inode information instead of block usage

**-l, –local :** limit listing to local file systems

**-P, –portability :** use POSIX output format

**–sync :** invoke sync before getting usage info

**–total :** elide all entries insignificant to available space, and produce grand total

**-t, –type=TYPE :** limit listing to file systems of type TYPE

**-T, –print-type :** print file system type

**du:** du command, short for disk usage, is used to estimate file space usage. The du command can be used to track the files and directories which are consuming excessive amount of space on hard disk drive.

**-0, –null :** end each output line with NULL

**-a, –all :** write count of all files, not just directories

**–apparent-size :** print apparent sizes, rather than disk usage.

**-B, –block-size=SIZE :** scale sizes to SIZE before printing on console

**-c, –total :** produce grand total

**-d, –max-depth=N :** print total for directory only if it is N or fewer levels below command line argument

**-h, –human-readable :** print sizes in human readable format

**-S, -separate-dirs :** for directories, don’t include size of subdirectories

**-s, –summarize :** display only total for each directory

**–time :** show time of last modification of any file or directory.

**–exclude=PATTERN :** exclude files that match PATTERN

**ulimit :**ulimit is admin access required Linux shell command which is used to see, set, or limit the resource usage of the current user. It is used to return the number of open file descriptors for each process. It is also used to set restrictions on the resources used by a process.

ulimit -u : To display maximum users process or for showing maximum user process limit for the logged-in user.

ulimit -f : maximum file size a user can have.

ulimit -m : maximum memory size for the current user.

**wc :**used to find out number of lines, word count, byte and characters count in the files specified in the file arguments.

* By default it displays four-columnar output.
* First column shows number of lines present in a file specified, second column shows number of words present in the file, third column shows number of characters present in file and fourth column itself is the file name which are given as argument.

wc state.txt with options -l: line, -w: word, -c: character

5 7 63 state.txt

**sort :** SORT command is used to sort a file, arranging the records in a particular order. By default, the sort command sorts file assuming the contents are ASCII. Using options in the sort command can also be used to sort numerically.

sort inputfile.txt: normal sort the contents of the file and print

sort -r inputfile.txt: reverse

sort -n inputfile.txt : numerically

-u: sort and remove duplicates

**cut :** The cut command in UNIX is a command for cutting out the sections from each line of files and writing the result to standard output.

List without ranges

$ cut -b 1,2,3 state.txt : only character positions

List with ranges

$cut -b 1-3,5-7 state.txt

**grep:** The grep filter searches a file for a particular pattern of characters, and displays all lines that contain that pattern

$grep -i "UNix" file.txt :Case insensitive search

$grep -c "unix" file.txt : count of number of matches

$grep -l "unix" \* :file names that matches the pattern

**ps:** Shows the processes for the current shell

PID – the unique process ID

TTY – terminal type that the user is logged into

TIME – amount of CPU in minutes and seconds that the process has been running

CMD – name of the command that launched the process.

ps -A :View all the running processes

ps -a :View all processes except both session leaders and processes not associated with a terminal.

ps -T :view all processes associated with this terminal

ps -r : View all the running processes :

ps -x : View all the processes owned by you:

**kill :i**s a built-in command which is used to terminate processes manually.

kill -l **:**To display all the available signals you can use below command option:

kill pid **:** To show how to use a *PID* with the *kill* command.

kill 6065

**nice:** nice command in Linux helps in execution of a program/process with modified scheduling priority

nice -10 gnome-terminal :+ve

nice --10 gnome-terminal :-ve

sudo renice -n 15 -p 77982 : changing the priority of running process

**echo :** echo command in linux is used to display line of text/string that are passed as an argument

echo “hello”

echo -e “hello \ hi” slash interpretation

**read:** used to read from a file descriptor

echo "what is your name..?";

read name;

echo "hello $name"

**head:** print the top N number of data of the given input. By default, it prints the first 10 lines of the specified files. If more than one file name is provided then data from each file is preceded by its file name.

head state.txt

head -n 5 state.txt

**tail:** print the last N number of data of the given input. By default it prints the last 10 lines of the specified files. If more than one file name is provided then data from each file is precedes by its file name.

tail state.txt

tail -n 5 state.txt

**awk**

**pg**

**more**

**pipe**

**tee**

**i/o redirection**

### Different ways of comparing two files in Unix

**#1) cmp: This command is used to compare two files character by character.**

* Syntax: cmp [options] file1 file2
* Example: Add write permission for user, group and others for file1.
  + *$ cmp file1 file2*

**#2) comm: This command is used to compare two sorted files.**

* Syntax: c*omm [options] file1 file2*
* One set of options allows selection of ‘columns’ to suppress.
  + -1: suppress lines unique to file1 (column 1)
  + -2: suppress lines unique to file2 (column 2)
  + -3: suppress lines common to file1 and file2 (column3)
* Example: Only show column-3 that contains lines common between file1 and file2
  + *$ comm -12 file1 file2*

**#3) diff: This command is used to compare two files line by line.**

* Description: The output indicates how the lines in each file are different, and the steps invoved to change file1 to file2. The ‘patch’ command can be used to make the suggested changes. The output is formatted as blocks of:

Change commands

*< lines from file1*

*—*

*> lines from file2*

The change commands are in the format [range][acd][range]. The range on the left may be a line number or a comma-separated range of line numbers referring to file1, and the range on the right similarly refers to file2. The character in the middle indicates the action i.e. add, change or delete.

* ‘LaR’ – Add lines in range ‘R’ from file2 after line ‘L’ in file1.
* ‘FcT’ – Change lines in range ‘F’ of file1 to lines in range ‘T’ of file2.
* ‘RdL’ – Delete lines in range ‘R’ from file1 that would have appeared at line ‘L’ in file2
* Syntax: *diff [options] file1 file2*
* Example: Add write permission for user, group and others for file1
  + *$ diff file1 file2*

**#4) dircmp: This command is used to compare the contents of directories.**

* Description: This command works on older versions of Unix. In order to compare the directories in the newer versions of Unix, we can use diff -r
* Syntax: *dircmp [options] dir1 dir2*
* Example: Compare contents of dir1 and dir2
  + *$ dircmp dir1 dir2*

**#5) uniq: This command is used to filter the repeated lines in a file which are adjacent to each other**

* Syntax: *uniq [options] [input [output]]*
* Example: Omit repeated lines which are adjacent to each other in file1 and print the repeated lines only once
  + *$ uniq file1*

**#!/bin/bash**

**file1="/home/vekomy/santhosh/bigfiles.txt"**

**file2="/home/vekomy/santhosh/bigfile2.txt"**

**if cmp -s "$file1" "$file2"; then**

**printf 'The file "%s" is the same as "%s"\n' "$file1" "$file2"**

**else**

**printf 'The file "%s" is different from "%s"\n' "$file1" "$file2"**

**fi**

**Combine multiple file in single file**

**#!/bin/sh**

**sort "$1" "$2" | uniq > "$3"**

**should do the trick. Sort will sort the concatenation of the two files (two first args of the script), pass the result to uniq which will remove adjacent identical lines and push the result into the third file (third arg of the script).**

**# Sort the input files directly with duplicates removed and save to output file.**

**sort -u "$1" "$2" > "$3"**

**#without sorting**

**cat $file1 $file2 $file3 >> $out**

## Understanding How Cron Works

Cron jobs are recorded and managed in a special file known as a crontab. Each user profile on the system can have their own crontab where they can schedule jobs, which is stored under /var/spool/cron/crontabs/.

To schedule a job, open up your crontab for editing and add a task written in the form of a *cron expression*. The syntax for cron expressions can be broken down into two elements: the schedule and the command to run.

The command can be virtually any command you would normally run on the command line. The schedule component of the syntax is broken down into 5 different fields, which are written in the following order:

|  |  |
| --- | --- |
| **Field** | **Allowed Values** |
| minute | 0-59 |
| hour | 0-23 |
| Day of the month | 1-31 |
| month | 1-12 or JAN-DEC |
| Day of the week | 0-6 or SUN-SAT |

Together, tasks scheduled in a crontab are structured like the following:

minute hour day\_of\_month month day\_of\_week command\_to\_run

Here’s a functional example of a cron expression. This expression runs the command curl http://www.google.com every Tuesday at 5:30 PM:

30 17 \* \* 2 curl http://www.google.com

There are also a few special characters you can include in the schedule component of a cron expression to streamline scheduling tasks:

* \*: In cron expressions, an asterisk is a wildcard variable that represents “all.” Thus, a task scheduled with \* \* \* \* \* ... will run every minute of every hour of every day of every month.
* ,: Commas break up scheduling values to form a list. If you want to have a task run at the beginning and middle of every hour, rather than writing out two separate tasks (e.g., 0 \* \* \* \* ... and 30 \* \* \* \* ...), you could achieve the same functionality with one (0,30 \* \* \* \* ...).
* -: A hyphen represents a range of values in the schedule field. Instead of having 30 separate scheduled tasks for a command you want to run for the first 30 minutes of every hour (as in 0 \* \* \* \* ..., 1 \* \* \* \* ..., 2 \* \* \* \* ..., and so on), instead, you could schedule it as 0-29 \* \* \* \* ....
* /: You can use a forward slash with an asterisk to express a step value. For example, instead of writing out eight separate separate cron tasks to run a command every three hours (as in, 0 0 \* \* \* ..., 0 3 \* \* \* ..., 0 6 \* \* \* ..., and so on), you could schedule it to run like this: 0 \*/3 \* \* \* ....

Here are some more examples of how to use cron’s scheduling component:

* \* \* \* \* \* - Run the command every minute.
* 12 \* \* \* \* - Run the command 12 minutes after every hour.
* 0,15,30,45 \* \* \* \* - Run the command every 15 minutes.
* \*/15 \* \* \* \* - Run the command every 15 minutes.
* 0 4 \* \* \* - Run the command every day at 4:00 AM.
* 0 4 \* \* 2-4 - Run the command every Tuesday, Wednesday, and Thursday at 4:00 AM.
* 20,40 \*/8 \* 7-12 \* - Run the command on the 20th and 40th minute of every 8th hour every day of the last 6 months of the year.