**LİNUX PLUS COMMANDS**

**Man ;** very structured documentation source **exp:**man ls

**Info ;**You can also read the Info pages about a command in addition to the man pages **exp:**info ls

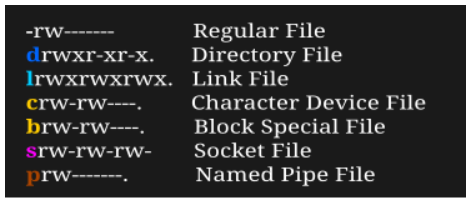
**whatis;**Displays brief description of a command **exp:**whatis ls

**apropos;**gives more information exp:apropos find

--help ;gives a short explanation about how to use the command exp:ls --help



**The standard scheme is in /etc/DIR\_COLORS:**

****

**Color\_File Type**

**blue \_directories**

**red\_**compressed archives

**white**\_text files

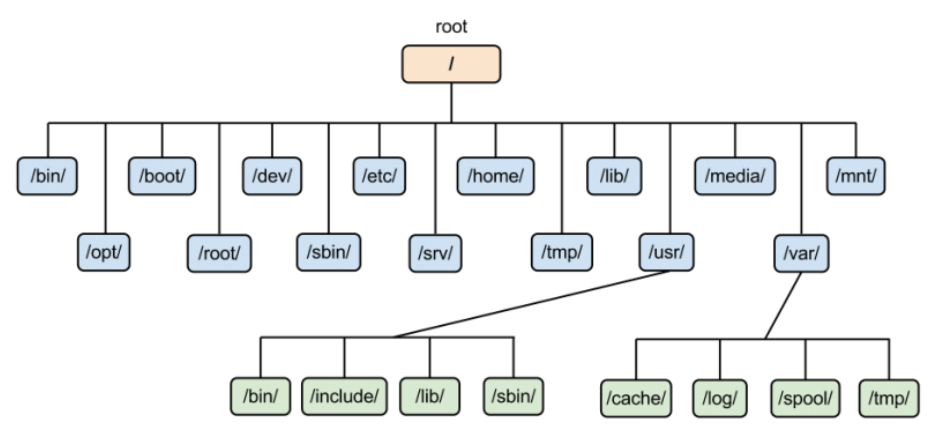
**pink\_** images

**cyan** \_links

**yellow**\_devices

**green**\_executables

**flashing red\_**broken links



**Common Environment Variables**

**Variable Description**

PATH This variable contains a colon (:)-separated list of directories in which your system looks for executable files.

USER The username

HOME Default path to the user's home directory

EDITOR Path to the program which edits the content of files

UID User's unique ID

TERM Default terminal emulator

SHELL Shell being used by the user

LANG The current locales settings.

**env** The command allows you to run another program in a custom environment without modifying the current one. When used without an argument it will print a list of the current environment variables.

**printenv** The command prints all or the specified environment variables.

**set**  The command sets or unsets shell variables. When used without an argument it will print a list of all variables including environment and shell variables, and shell functions.

**unset**  The command deletes shell and environment variables.

**export** The command sets environment variables.

**Command Description**

**echo $VARIABLE**  To display value of a variable

**env**  Displays all environment variables

**VARIABLE\_NAME= variable\_value** Create a new variable

**echo $VARIABLE**  To display value of a variable

**unset**  Remove a variable

**export Variable=value** To set value of an environment variable

**echo $PATH :**

/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games

Add a New Directory to the Path ; $ export PATH=$PATH:/games/awesome

**USERS AND GROUPS**

**sudo ;** (superuser do) command gives some admin privileges to non-admin users

**Commands Meaning**

|  |  |
| --- | --- |
| **sudo -l** | List available commands. |
| **sudo command** | Run command as root. |
| **sudo -u root command** | Run command as root. |
| **sudo -u user command** | Run command as user. |
| **sudo su** | Switch to the superuser account. |
| **sudo su -** | Switch to the superuser account with root's environment. |
| **sudo su - username** | Switch to the username's account with the username's environment. |
| **sudo -s** | Start a shell as root |
| **sudo -u root -s** | Same as above. |
| **sudo -u user -s** | Start a shell as user. |

**su ;** enables a shell to be run as another user.

**su - ;** retains the same shell environment by default.

**Whoami ;** Your username is indicated .

**Who ;** will provide you with details about who is logged on the system

**who am i ;** will only show the line that points to your current session.

**w ;** will inform you who is logged on and what they are doing.

**İd ;**will provide your user id, your primary group id, and a list of the groups you belong to.

/etc/passwd; On Linux, the local user database is /etc/passwd.

**root;** The most powerful account on your Linux system is the root user also called the superuser. This user is capable of doing almost everything, even creating other users. The userid of the root user is always 0.

root@DESKTOP-UN6T2ES:~# id

uid=0(root) gid=0(root) groups=0(root)

root@DESKTOP-UN6T2ES:~#

**useradd;**  used for creating a new user.

**exp:** useradd -m -d /home/walter -c "walter clarus" walter

* -m is used for forcing the creation of the home directory
* -d is used for setting the name of the home directory
* -c is used for setting a description

**adduser ;** basically a [Perl](https://www.perl.org/) script that uses the useradd command in the background. This is more effective at creating new users on Linux.**userdel;** You can delete a user. If you want to remove the home directory, you need to use the-r in the command line**. exp**: userdel -r James

**usermod;** you can modify a user's properties.

**exp:** usermod -c 'aws solution architect' walter

**passwd;** User passwords can be set

**exp:** oliver@DESKTOP-UN6T2ES:~$ passwd / root@DESKTOP-UN6T2ES:~# passwd john

**shadow file:** User passwords are encrypted and stored in /etc / shadow file. The /etc/shadow file is only read and can be accessed by root only.

**/etc/login.defs;** includes some default user password settings, such as password aging and length settings.

**GROUP MANAGEMENT**

**groupadd;** used to create a new group. Exp: groupadd linux

**usermod;** You can change group membership with the useradd or usermod command.

root@DESKTOP-UN6T2ES:~# usermod -a -G linux James

**Group File;** Users can belong to several groups. Group membership is specified via the /etc root@DESKTOP-UN6T2ES:/home/clarusway# tail -3 /etc/group

linux:x:1006:john,james,aaron

aws:x:1007:walter

python:x:1008:oliver/group file.

**groups;** used to display a list of groups to which the user belongs.

**groupmod;** command can be used to change the group name. Exp: groupmod -n ubuntu linux

**groupdel** ;command is used to delete a group. Exp: groupdel Ubuntu

**gpasswd ;** we can add a user to a group and to remove a user from a group.

We add john to aws group with **gpasswd -a john aws**

We remove walter from aws group with **gpasswd -d walter aws**

**YUM (Yellowdog Updater Modified)**

YUM is an open-source package manager that was developed by Duke University. It is used both in the command line and GUI. It supports numerous repositories. It works mostly the same as APT in Debian Linux systems. Here are some examples of YUM.

$ yum install # Installing a package

$ yum -y install # During installment, linux asks for confirmation. To skip confirmation you can use option -y.

$ yum remove # Removing a package with all dependencies.

$ yum update # Updating a package

**cat;** When between two pipes, the cat command does nothing (except putting stdin on stdout). Displays the text of the file line by line.

user@clarusway:~$ tac count.txt | cat | cat | cat | cat | cat

five

four

three

two

one

**tee ;** almost the same as cat, except that it has two identical outputs.

user@clarusway:~$ tac count.txt | tee temp.txt | tac

one

two

three

four

five

user@clarusway:~$ cat temp.txt

five

four

three

two

one

**grep ;** The most common use of grep is to filter lines of text containing (or not containing) a certain string.

user@clarusway:~$ cat tennis.txt | grep Williams

Serena Williams, USA

Venus Williams, USA

**cut ;** can select columns from files, depending on a delimiter or a count of bytes.

clarusway@DESKTOP-UN6T2ES:~$ cut -d: -f1-3 /etc/passwd | tail -3

d means delimiter. In the example above, colon (:) is used as a delimiter.

f means field.

**tr ;** It is used for translating and deleting characters./To delete the characters.

cat clarusway.txt | tr -d e / cat clarusway.txt | tr [a-z] [A-Z] (To convert lower case to upper case) / cat clarusway.txt | tr [:space:] '\t' (To translate white-space to tabs)

**wc;** Counting words, lines and characters is easy with w.. wc -l tennis.txt wc -w wc -c

**sort ;** he sort filter will default to an alphabetical sort**. Exp :** **sort music.txt**

sort -r the flag returns the results in reverse order

sort -f the flag does case insensitive sorting

sort -n the flag returns the results as per numerical order

**uniq;** You can remove duplicates from a sorted list**.Exp: sort music.txt |uniq**

**comm;** Comparing streams (or files) can be done with the comm

$ cat list1.txt / $ cat list2.txt/ $ comm list1.txt list2.txt

**OPERATORS**

**Control Operator Usage**

**; semicolon** More than one command can be used in a single line.

**& ampersand** Command ends with & and doesn't wait for the commandto finish.

**$? dollar question mark** Used to store exit code of the previous **command.**

**&& double ampersand** Used as logical AND.

**|| double vertical bar** Used as logical OR.

**Combining && and ||** Used to write if then else structure in the command line.

**# pound sign** Anything was written after # will be ignored.

mkdir test # we create a directory

**combining && and ||;** **Y**ou can use this logical AND and logical OR to write an if-then-else structure on the command line. This example uses echo to display whether the rm command was successful.

**rm file1 && echo It worked! || echo It failed!**

**\ escaping special characters and end of line backslash ;** The backslash \ character enables the use of control characters, but without the shell interpreting it, this is called escaping characters.

Lines ending in a backslash are continued on the next line

user@clarusway:~$ echo This command line \

> is split in three \

> parts

This command line is split in three parts

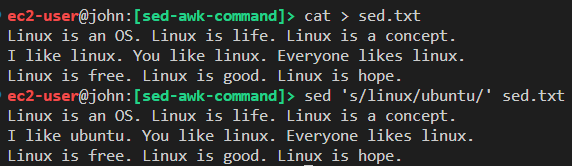
**sed (command**) A stream editor. A stream editor is used to perform a lot of function on a file like searching, find and replace, insertion or deletion.

**### Replacing or substituting string**

Linux is an OS. Linux is life. Linux is a concept.

I like linux. You like linux. Everyone likes linux.

Linux is free. Linux is good. Linux is hope.



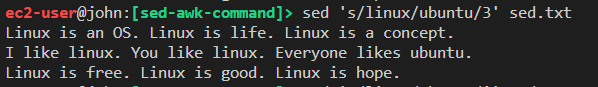
`s` specifies the substitution operation.

- The `/` are delimiters.

- The `linux` is the search pattern and the `ubuntu` is the replacement string.

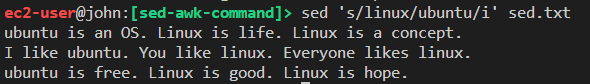
**### Replacing the any occurrence of a pattern in a line**

Use the /1, /2 etc flags to replace the first, second occurrence of a pattern in a line. The following command replaces the third occurrence of the word “linux” with “ubuntu” in a line.



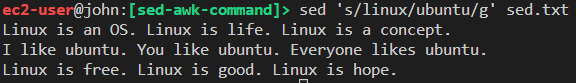
**### Replacing a string by ignoring case distinctions.**

By, default sed command do not ignore case distinctions. For this `i` pattern can be used.

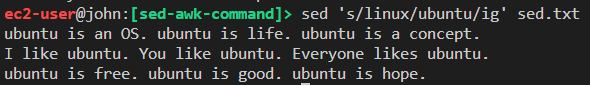


**#### Replacing all the occurrence of the pattern in a line**

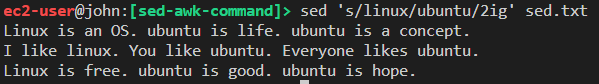
`g flag` (global replacement) defines the sed command to replace all the occurrences of the string in the line.



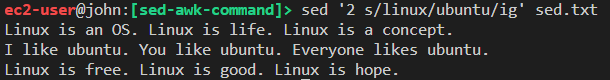
We can do the same by ignoring case distinctions. Use the combination of `/i` and `/g`.



We can replace all the patterns from the any occurrence of a pattern in a line by using the combination of /1, /2 etc and /g. The sed command below replaces the second, third, and so on “linux” word with “ubuntu” word in a line.



We can limit the sed command to replace the string on a specific line number. The following command only replaces the second line.



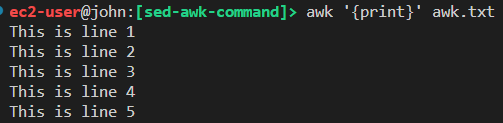
**awk (command)**

a text pattern scanning and processing language. It searches one or more files to see if they contain lines that matches with the specified patterns and then performs the associated actions.

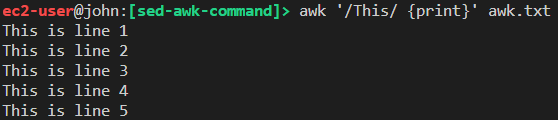
- While the sed program works well with character-based processing, the awk program works well with delimited field processing.

awk options 'selection \_criteria {action }' file

- By default Awk prints every line of data from the specified file.



**### Print the lines which matches with the given pattern**



**### Splitting a Line Into Fields**

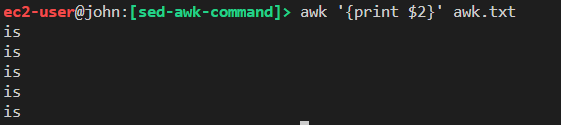
By default, the awk command splits the record delimited by a whitespace character.  Awk assigns some variables for each data field as below:

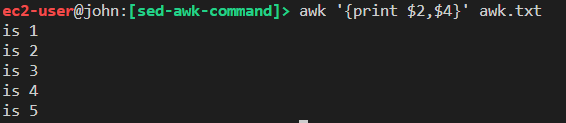
$0 for the whole line.

$1 for the first field.

$2 for the second field.

$n for the nth field.





We can change delimiter by using –F option. First, update the awk.txt and Let's separate the fields by `:` as below.

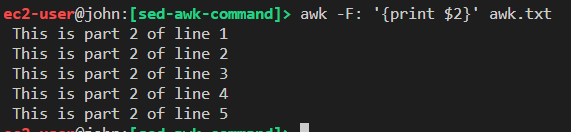
This is part 1 of line 1 : This is part 2 of line 1

This is part 1 of line 2 : This is part 2 of line 2

This is part 1 of line 3 : This is part 2 of line 3

This is part 1 of line 4 : This is part 2 of line 4

This is part 1 of line 5 : This is part 2 of line 5



We can use awk command as filter.

ls -l | awk '{print $9}'

We can find any string in any specific column.

awk '{ if($7 == "3") print $0;}' awk.txt

**crontab ;** stands for `cron table`, which is a list of commands scheduled to run at regular time intervals on the system.

- If we need to schedule any task on Linux, we should basically edit the crontab file. We can do that using the below command.

crontab -e               # edit the crontab file

crontab -l               # list current cron tasks

crontab -r    # Remove your crontab file

crontab -u username -e # edit other users's crontab file

Editing the crontab file is not complex, but we should first learn how to set a date and time using 5 \* on that file. There are six fields that we use on every cron task line. Those are explained in detail in the below picture.

\* \* \* \* \* <shell command>   # execute cron job every minute

0 1 \* \* \* <shell command>   # execute cron job every day at 1 a.m.

\* \* 1 \* \* <shell command>   # execute every minute in January

\* \* \* \* 6 <shell command>   # execute every minute on every saturday

0 1/15 \* jan,jun mon,fri <command> # execute at every 1 a.m. and 3

                                     p.m. every monday and friday on

                                     january and june

<https://crontab.guru>

We can also use some regular expressions to define the date part.

\* = Any/All values           # e.g. \*

- = Range of values          # e.g. 1-5

, = Multiple/List of values  # e.g. 1,2,3

/ = Step values               # e.g. 1/3

Finally let’s create some crontab tasks. Create a cron task writes the system date information every day at 1 p.m. to the date.log file.

```bash

crontab -e

0 13 \* \* \* date >> /home/ec2-user/date.log```

- Create a cron task updates and upgrades our server every Sunday at 3 a.m.

```bash

0 3 \* \* sun sudo yum update -y```

-  List the cron tasks.

```bash

crontab -l

- basic regular expressions:

| Symbol| Descriptions |

| -------- | ----------- |

| . | replaces any character |

| ^ | matches start of string |

| $ | matches end of string |

| \* | matches up zero or more times the preceding character |

| \ | Represent special characters |

| () | Groups regular expressions |

| ? | Matches up exactly one character |

- `{n}`, matches the preceding character appearing ‘n’ times exactly

Cat fruits.txt | grep k.k

cat fruits.txt | grep ^b

cat fruits.txt | grep n$

cat fruits.txt | grep k[adb]k

cat fruits.txt | grep k[a-z]k

cat fruits.txt | grep k[\*]k

cat fruits.txt | grep k[A-Z]k

cat fruits.txt | grep k[a-zA-Z]k

cat fruits.txt | grep k[0-9]k

cat fruits.txt | grep k[a-zA-Z0-9]k

cat fruits.txt | grep -E “p{2}” #-E ; PATTERN is an extended regular express