COMANDOS DE LINUX

Todos los comandos mencionados tienen parámetros que no menciono acá, pero que permiten muchas funcionalidades muy útiles.
Para darle privilegios de sudo a un usuario en GNU/Linux, necesitas agregarlo al grupo sudo:
sudo usermod -aG sudo (nombre_usuario)
man (comando de la Shell): muestra, si es que tiene un manual del comando.
help: muestra información de los comandos internos.

ls -l: lista todos los archivos del directorio en el que estés y muestra los permisos.
El primer carácter indica el tipo de archivo:

- **-**: Archivo regular.
- **d**: Directorio.
- **l**: Enlace simbólico.
- **b, c**: Archivos especiales de dispositivo.

./(archivo ejecutable): permite ejecutar un archivo ejecutable por cualquier intérprete que esté disponible en el sistema. (un script de python por ejemplo)

sudo (comando): permite actuar como superusuario o usuario root mientras ejecutas un comando específico.

touch (nombre): crea un archivo vacío o también actualiza el tiempo de acceso de un archivo.

rm (archivo): elimina un archivo o directorio. (cuidado) (antes del archivo, agregar **-r** para directorios vacíos y **-rf** para directorios con algo dentro, es para forzarlo)

cat (nombre): muestra un archivo, si no existe lo crea **(cat > nombre)**, concatena archivos. **(cat arch1 arch2 > nuevoarch)**

zcat (archivo_comprimido): idéntico al comando cat, pero para archivos comprimidos .gz.

more (archivo): muestra un archivo una página a la vez. (es mejor que el cat para visualizar mejor un archivo)

mkdir (nombre): crea un directorio.

pwd: visualiza la ruta en la que estoy parado.

cd (archivo): acceder a un archivo. (**cd ~** accede directamente al directorio personal del usuario actual)

find (directorio) -name “cadena*”: busca todas las coincidencias de una cadena de caracteres.
ejemplo: **find /home -type f -name "*.conf"** (busca en /home todos los archivos que terminen con .conf, solo archivos no directorios, si quiero directorios también, sacar el -type f)

df: informar la cantidad de espacio libre en disco.

who: verifique los usuarios conectados al sistema.

tail: muestra las últimas líneas de un archivo.

shutdown (-h / -r) now: apaga (h) o reinicia (r) el sistema. (también se puede programar una hora específica)

reboot: reinicia el sistema inmediatamente.

halt: detiene el sistema inmediatamente.
locate -i (archivo): busca archivos en el sistema basándose en una base de datos actualizada. (-i significa que ignora mayúsculas y minúsculas) (es necesario actualizar la base de datos para que busque correctamente, ejecutar: **sudo updatedb**)
uname -a: muestra información del sistema.

dmesg: muestra mensajes del buffer del núcleo (kernel), generalmente mensajes de arranque y del sistema.

lspci: lista todos los dispositivos PCI conectados al sistema. (periféricos conectados a la placa madre)

at (hora): permite programar comandos a una hora en específica.

netstat: muestra información sobre conexiones de red, tablas de enrutamiento, interfaces, etc.

mount: monta un sistema de archivos.

unmount: desmonta un sistema de archivos.

head -n (número) (archivo): muestra las primeras líneas de un archivo.

losetup: configura y controla dispositivos de bucle. (loopback)

write: permite enviar mensajes a otros usuarios en el mismo sistema.

mkfs: se utiliza para crear un sistema de archivos en una partición o disco.

fdisk: se utiliza para manipular las tablas de particiones.

useradd -m -s (nombre usuario): agrega un usuario al sistema. (el parámetro -m le crea un /home y el -s permite elegir declarar la shell del usuario, son opcionales)

sudo usermod -a -G (nombre_grupo) (nombre_usuario): agrega un usuario a un grupo. (el parámetro -a hace que lo agregue al grupo sin eliminarlo de los grupos en los que ya este, y -G permite especificar los grupos a los que el usuario debe ser agregado)

sudo groupadd (nuevo_grupo): crea un nuevo grupo

sudo userdel -r (nombre_usuario): elimina a un usuario, su home y correo, útil para eliminar todo rastro de un usuario. (el parámetro -r es el que elimina el home y correo)

getent group (nombre): retorna los usuarios que pertenecen al grupo indicado, junto con su información (usarlo con cut -d: -f4 para obtener solo los nombre)
getent passwd: lista los usuarios del sistema

su (nombre_usuario): permite cambiar de usuario, entre ellos al root en la sesión de la terminal actual.

sudo groupdel (nombre_grupo): elimina un grupo del sistema.

passwd: permite cambiar tu propia contraseña.

sudo passwd (nombre_usuario): permite cambiar la contraseña de otros usuarios. (si se le agrega el parámetro -d elimina la contraseña, y permite ingresar al usuario sin una contraseña)

chmod (parametros)(archivo): permite cambiar los permisos de archivos y directorios. Este comando permite controlar quién puede leer, escribir o ejecutar un archivo o acceder a un directorio, asignando los permisos adecuados para el propietario, el grupo y otros usuarios.
Se suman estos valores para definir los permisos de cada categoría:

- 0: Sin permisos.
- 1: Sólo ejecución.
- 2: Sólo escritura.
- 4: Solo lectura.
- 5: Lectura y ejecución.
- 6: Lectura y escritura.
- 7: Lectura, escritura y ejecución.

Ejemplo: **chmod 777 archivo.txt**, le da permisos completos (lectura, escritura y ejecución) para todos.

sudo chown (parámetros) (propietario)(:grupo) (archivo_o_directorio): permite cambiar el propietario y/o el grupo al que pertenece un archivo o directorio.
Parámetros:
-R: especifica que todos los archivos y directorios dentro del archivo a cambiar de dueño también cambien de dueño.

chgrp (parámetros) (grupo) (archivo_o_directorio): permite cambiar el grupo al que pertenece un archivo o directorio.

umount (parámetros) (punto_de_montaje o dispositivo): se utiliza para desmontar un sistema de archivos o una partición que ha sido previamente montada.

du (archivo): muestra el uso del espacio en disco de archivos y directorios.

rmdir (directorio): se utiliza para eliminar específicamente directorios vacíos.

df (directorio): sirve para mostrar información sobre el espacio disponible y utilizado en los sistemas de archivos montados en el sistema.

ln (archivo_ya_creado) (nombre_nuevo_archivo): se utiliza para crear enlaces entre archivos. Un enlace es una referencia o puntero a otro archivo en el sistema de archivos. (los cambios hechos en el archivo ya creado se verán reflejados en el nuevo archivo) (posee otros parámetros)

ls: muestra todos los archivos y directorios de donde estes parado.

cp (origen) (destino): se utiliza para copiar archivos y directorios de un lugar a otro.

mv “archivo_a_renombrar” “directorio/nuevo_nombre”: se utiliza para mover o renombrar archivos y directorios. (investigar cómo aplicar las otras funcionalidad)

ps: muestra una instantánea de los procesos en ejecución con detalle en el momento en que se ejecuta. (parámetros: **-e** / **-ef** / **-e –forest** / **-aux** (muestra con detalles)).

top: muestra una vista dinámica y en tiempo real de los procesos que están utilizando el sistema.

jobs: ver procesos en background.
fg %(JID): pasas un proceso de background a foreground.
bg %(JID): pasas un proceso de foreground a background. (primero hay que detener el proceso con ctrl + z)

pstree: muestra los procesos en ejecución en forma de un árbol jerárquico.
killall (nombre_programa): termina todos los procesos que coincidan con un nombre específico de programa.

nice (valor_de_nice) (programa): permite establecer la prioridad de ejecución de un proceso. Cuanto mayor sea el valor de `nice`, menos prioridad tendrá el proceso, permitiendo que otros procesos tengan más acceso a la CPU.
El valor de `nice` puede estar entre **-20 (máxima prioridad)** y **19 (mínima prioridad)**. Por defecto, los procesos se ejecutan con un valor de nice de **0**.

- **Valores negativos**: Prioridad más alta (consume más CPU).
- **Valores positivos**: Prioridad más baja (consume menos CPU).

tar (parametros) (archivo_tar) (archivos_o_directorios_a_empaquetar): permite empaquetar archivos y extraer archivos empaquetados en un archivo de tipo tarball (archivo `.tar`).

Parámetros comunes

- `-c`: Crear un nuevo archivo tar.
- `-x`: Extraer archivos de un archivo tar.
- `-v`: Verbose, muestra los archivos procesados durante la operación.
- `-f`: Indica el archivo tar con el que trabajar.
- `-z`: Comprimir o descomprimir usando **gzip**.
- `-j`: Comprimir o descomprimir usando **bzip2**.
- `-t`: Mostrar el contenido de un archivo tar sin extraerlo.

Ejemplo: para empaquetar y comprimir usando gzip.
> tar -czvf archivo.tar.gz archivo1 archivo2 directorio/

grep (parámetros) (“patron_de_busqueda”) (archivo(s)): se utiliza para **buscar patrones específicos de texto** dentro de archivos o en la salida de otros comandos.
Parámetros:
-i: ignora mayúsculas y minúsculas.
-n: muestra el número de línea donde se encuentra el texto coincidente.
-r: busca recursivamente en los directorios.
-w: para buscar solo la palabra exacta indicada.
-v: invertir la búsqueda. Retorna las líneas que **no** coincidan con el patrón.
-c: cuenta las coincidencias del patrón en las líneas del texto.
grep “^patrón”: busca las líneas que contengan el patrón al inicio de la línea.
grep “patrón$”: busca las líneas que contengan el patrón al final de la línea.

zgrep (parámetros) (“patron_de_busqueda”) (archivo(s)): idéntico al comando grep, pero para archivos comprimidos .gz.

gzip (parámetros) (archivo(s)): se utiliza para **comprimir y descomprimir archivos**.
Parámetros:
-k: mantiene los archivos originales. (si no se declara, los archivos originales se eliminan)
-d: descomprime un archivo .gz.

wc (parámetros) (archivo(s)): se utiliza para **contar líneas, palabras y caracteres** en archivos de texto.

COMANDOS BASH

Introducción a bash:

Variables: Strings y Arreglos().
#... :Comentarios.
Estructuras de control: if, while, for (2 tipos) y case.
echo (“...”): imprimir texto.
echo -e “asd\tasd\tasd\t”: permite tabular entre el texto
Ejemplo:

```
# Crea o sobrescribe el archivo reporte.txt
echo -e "Nombre de Usuario\tCantidad de Archivos" > reporte.txt
echo -e "$USER\t\t$COUNT" >> reporte.txt
la opción \n salta de línea.
```

read var: leer una línea desde entrada estándar en la variable var.
cut -d: -f1 “string”: Sirve para extraer columnas específicas de un texto.

comando_que_retorne_string | cut -d: -f1
-d'delimitador': Permite especificar el tipo de delimitador (por defecto cut usa la tabulación)
-f1,2,3,4, … : Especifica las columnas a extraer

wc -l: Contar las líneas que se leen desde entrada estándar.
Los procesos (programas en ejecución) normalmente cuentan con 3 "archivos" abiertos.

- **stdin:** Entrada estándar, normalmente el teclado. (0)
- **stdout:** Salida estándar, normalmente el monitor. (1)
- **stderr:** Error estándar, normalmente la salida estándar. (2)

• **>** Redirección destructiva: Si el archivo no existe, se crea. Si el archivo existe, sobreescribe.
• **>>** Redirección no destructiva: • Si el archivo no existe, se crea. Si el archivo existe, se agrega al final.
• **2> y 2>>** Redirigen el error estándar.
• **comando < archivo** Hace que archivo sea la entrada de comando. En otras palabras cuando comandó intente leer entrada del teclado, en realidad, va a leer el contenido de archivo.
|: Pipes, conecta la salida de un comando con otro.

Variables:

- Son inicializadas por defecto con valor nulo o 0 dependiendo del contexto de evaluación.
- Por defecto las variables son globales
- A una variable local dentro de una función se le antepone **local**:

```
test() {
        local variable
}
```

- Las variables de entorno son heredadas por los procesos hijos.
- Para exponer una variable global a los procesos hijos se usa el comando export:

```
export VARIABLE_GLOBAL="Mi var global" comando
# comando vera entre sus variables de entorno a VARIABLE_GLOBAL
```

Declarar una variable: **NOMBRE="asd"** (no debe haber espacios entre él "=" y no pueden empezar con un número)
Acceder a una variable:
$VARIABLE

- Para evitar ambigüedades se pueden usar llaves:

```
# Esto no accede a $NOMBRE.
$NOMBREesto_no_es_parte_de_la_variable
# Esto si.
${NOMBRE}esto_no_es_parte_de_la_variable
```

Arreglos:

Creación:

```
arreglo_a=() # Se crea vacío
arreglo_b=(1 2 3 5 8 13 21) # Inicializado
```

Asignación de un valor en una posición concreta:

```
arreglo_b[2]=spam
```

Acceso a un valor del arreglo (En este caso las llaves no son opcionales):

```
${arreglo_b[2]}
copia=${arreglo_b[2]}
```

Acceso a todos los valores del arreglo: permite imprimir todos los valores

Acceder de esta forma para el foreach en bash (el del dibujo no sirve)

```
${arreglo[@]} # o bien ${arreglo[*]}
```

Tamaño del arreglo:

```
${#arreglo[@]} # o bien ${#arreglo[*]}
```

Borrado de un elemento (reduce el tamaño del arreglo pero no elimina la posición, solamente la deja vaca):

```
unset arreglo[2]
```

Agregar al final del arreglo:

```
arreglo+=(“string”) o arreglo+=(5)
```

Se puede inicializar un arreglo con el resultado de un comando:
arreglo=($(cat /etc/passwd | cut -d: -f1)) es un ejemplo de inicializar un arreglo con los nombres de usuarios en el sistema.

Los índices en los arreglos comienzan en 0.

Variables y comillas:

No hacen falta, a menos que:

1. el string tenga espacios.
2. que sea una variable cuyo contenido pueda tener espacios.
3. son importantes en las condiciones de los if, while, etc…

Las comillas dobles permiten usar variables dentro:

```
echo "El resultado es: $(ls)"
```

Reemplazo de comandos:

- Permite utilizar la salida de un comando como si fuese una cadena de texto normal.
- Se la puede utilizar de dos formas, cada una con distintas reglas:
  $(comando_valido) (es mejor en general)
  `comando_valido`
  NOMBRE_VAR="$(comando_valido)"

Ejecutar un programa bash: bash archivo.sh

Selección de alternativas:

Decisión:

```
if [ condition ]
then
        block
fi
```

Selección:

```
case $variable in
        "valor 1")
                block
        ;;
        "valor 2")
                block
        ;;
        *)
                block
        ;;
esac
```

Menú de opciones:

```
select variable in opcion1 opcion2 opcion3
do
        # en $variable está el valor elegido
         block
done
```

Menú de opciones:

Ejemplo:

```
select action in New Exit
do
case $action in
        "New")
                echo "
                    Selected
                     option
                    is NEW"
        ;;
        "Exit")
                exit 0
        ;;
        esac
done
```

Imprime:

y espera el número de opción por teclado

- C-*style*:

```
for ((i=0; i < 10; i++))
do
        block
done
```

- Con lista de valores (`foreach`):

```
for i in value1 value2 value3 valueN;
do
        block
done
```

while

```
while [ condition ] #Mientras se cumpla la condición
do
        block
done
```

until

```
until [ condition ] #Mientras NO se cumpla la condición
do
        block
done
```

Ejemplos:

```
if [ "$USER" == root ]
then
        echo "super user"
else
        echo "Ud. es $USER"
fi
#-------------------------
n=0
while [ $n -ne 5 ]; do
        echo $n
        let n++
done
```

```
for archivo in $(ls)
do
        echo "- $archivo"
done
#-------------------------
for ((i=0; i<5; i++))
do
        echo $i
done
```

Adicionalmente:

- `break [n]` corta la ejecución de n niveles de *loops*.
- `continue [n]` salta a la siguiente iteración del enésimo *loop* que contiene esta instrucción.

```
# AND
if [ $a = $b ] && [ $a = $c ]
then
#...
# OR
if [ $a = $b ] || [ $a = $c ]
then
#...
```

Argumentos y valor de retorno:

- Los scripts pueden recibir argumentos en su invocación.
- Para accederlos, se utilizan variables especiales:

$0 contiene la invocación al script.
$1, $2, $3, ... contienen cada uno de los argumentos.
$# Contiene la cantidad de argumentos recibidos.
$* Contiene la lista de todos los argumentos.
$? Contiene en todo momento el valor de retorno del último comando ejecutado.
: Representa todos los archivos en el directorio actual en el que se ejecuto el script.

Terminación de un script:

exit

• Causa la terminación de un script
• Puede devolver cualquier valor entre 0 y 255:
• El valor 0 indica que el script se ejecutó de forma exitosa
• Un valor distinto indica un código de error
• Se puede consultar el exit status imprimiendo la variable $?

Funciones:

- Se pueden declarar de 2 formas:

function nombre { block }
nombre() { block }

- Con la sentencia return se retorna un valor entre 0 y 255
- El valor de retorno se puede evaluar mediante la variable $?
- Reciben argumentos en las variables $1, $2, etc

```
# Recibe 2 argumentos y devuelve:
#    1 si el primero es el mayor
#    0 en caso contrario
mayor()
{
        echo "Se van a comparar los valores $*"
        if [ $1 -gt $2 ]; then
                echo "$1 es el mayor"
                return 1
        fi
        echo "$2 es el mayor"
        return 0
}
mayor 5 6 # Invocación
echo $?   # Imprime el exit Status de la funcion
```

$HOME: contiene el directorio personal del usuario que ejecute el script

expr (expresión): se utiliza para **evaluar expresiones** y realizar operaciones aritméticas, operaciones de cadenas, o comparaciones lógicas.
> **expr $var_o_comand \| $var_o_comand** (or)
> **expr $var_o_comand \& $var_o_comand** (and)
> **expr length string** (largo de una cadena)

> **expr substr string pos_inicial cant_car** (a partir de la pos inicial, en adelante extraerá la cantidad de caracteres especificado)
> **expr $par/var * $par/var** (multiplicación)
> **expr $par/var / $par/var** (división)
> **expr $par/var + $par/var** (suma)
> **expr $par/var - $par/var** (resta)
> **expr $par/var % $par/var** (modulo - resta de la división) (su resultado también se interpreta como un true o false, por ejemplo: si es divisible por…)
> **expr $par/var \< $par/var** (menor que, retorna 1 si es verdadero y 0 si es falso)
> **expr $par/var \> $par/var** (mayor que, retorna 1 si es verdadero y 0 si es falso)
> **expr $par/var = $par/var** (igual a)

test expresión o **[expresión]**: boolean en programación. Retorna **0** si es verdadero o **1** si es falso. (los [ ] necesitan esos espacios)
Operadores para las comparaciones:

Operadores para condition:

Operador	Con strings	Con números
Igualdad	`"$nombre" = "Maria"`	`$edad -eq 20`
Desigualdad	`"$nombre" != "Maria"`	`$edad -ne 20`
Mayor	`A > Z`	`5 -gt 20`
Mayor o igual	`A >= Z`	`5 -ge 20`
Menor	`A < Z`	`5 -lt 20`
Menor o igual	`A <= Z`	`5 -le 20`

También:
> **-z**: la cadena está vacía.
> **-n**: la cadena no está vacía.
Pruebas de archivos: también permite verificar la existencia y propiedades de archivos.
[-e “archivo”]: verifica si el archivo existe.
[-f “archivo”]: verifica si es un archivo regular.
[-d “archivo”]: verifica si es un directorio.
[-r “archivo”]: verifica si el archivo es legible.
[-w “archivo”]: verifica si el archivo es escribible.
[-x “archivo”]: verifica si el archivo es ejecutable.

$(basename “$archivo”): Devuelve el nombre del archivo, sin la ruta. Útil para renombrar una colección de archivos.

“string” | tr (opciones) SET 1 (SET2): En el set 1 va lo que se va a cambiar o eliminar y en el set 2 su reemplazo.
“string” | tr -d set 1: En set 1 va lo que se va a eliminar.
“string” | tr “A-Z” “a-z”: pasa todo a minúscula y viceversa para pasar a mayúscula.
echo “archivo” | tr …: Usar echo para referirse a archivos
let “expresión”: sirve para hacer operaciones aritméticas fácilmente.
let “var–”: decrementa.
let “var++”: incrementa.
let “resultado = a + b”: suma las var a y b, siendo variables ya declaradas.

Es mejor usar (($var * 2)) para operaciones aritméticas.
Es mejor usar [["$var1" = “xd” && "$var2" != “xdnt”]] para operaciones lógicas, permite usar && y ||

Separador interno de los bucles: permite definir un separador para los datos iterador por un bucle, útil para agregar a una colección palabras de un string.
IFS=’separador’

Iterar sobre las rutas de los archivos de un directorio:
for archivo in “$directorio”/*; do

Es posible buscar los que tengan X extensión…
for archivo in “$directorio”/*.doc; do

shift [n]: permite desplazar n veces a la izquierda los parámetros enviados a un script y/o a una función, los desplaza n veces y los que estén más a la izquierda se eliminan.

date: obtiene fecha y hora actual (Fri Oct 18 14:50:09 UTC 2024).
>Se puede formatear **date +"%Y-%m-%d"** (retornara: 2024-10-18)

Para iterar sobre las líneas de un archivo usar cat en un for y combinarlo con pipes para ir filtrando lo que quiera:
for linea in $(cat /etc/passwd | cut -d: -f1,6); do por ejemplo me dará los nombres y directorios personales de cada usuario en el sistema, habrá uno por línea y los iterara.

COMANDOS PARA PROCESOS

top: Muestra en tiempo real el estado de los procesos en ejecución en el sistema, ordenados por consumo.

htop: Versión mejorada de top con una interfaz más amigable e interactiva. Navegación con teclas para seleccionar y finalizar procesos. Requiere instalación en algunas distribuciones.

ps: Muestra información sobre procesos en ejecución. Es estático.

pstree: Muestra los procesos del sistema en forma de árbol, destacando la jerarquía de padres e hijos.

kill (parámetros) (PID): sirve para enviar diferentes tipos de señales a los procesos para realizar acciones como detener, reiniciar o continuar la ejecución de un proceso.
Principales señales que puede enviar kill:
Terminar y reiniciar el proceso.

1. Interrupción (similar a Ctrl+C).
2. Salida desde el terminal.
3. Forzar la terminación inmediata.
4. Solicitud de terminación (por defecto).
5. Pausar (detener) el proceso.
6. Continuar la ejecución de un proceso detenido.

Ejemplo: Forzar la terminación de un proceso con PID = 1234.
> **kill -9 1234**

pgrep (nombre/patrón): Busca procesos basados en su nombre o patrones y todos los PID relacionados a ese proceso. (pgrep firefox)

killall (nombre/patrón): Finaliza todos los procesos con un nombre específico.
renice (valor) <PID>: Cambia la prioridad de un proceso ya en ejecución. Valores: -20 (mayor prioridad) a 19 (menor prioridad).

xkill: Herramienta gráfica para cerrar ventanas o aplicaciones en entornos gráficos. Requiere entorno gráfico y a veces instalación (sudo apt install x11-utils).

atop: Monitor de rendimiento avanzado que registra datos del sistema. Requiere instalación (sudo apt install atop).
fork(): Sirve para crear un nuevo proceso hijo duplicando el proceso actual (proceso padre).

- Todo lo que sucede después de una llamada a `fork()` **se ejecutará en cada proceso creado** (tanto el proceso padre como los hijos)(cada proceso tiene su propio espacio de memoria, almacenan sus propias variables, no las comparten entre procesos).
- A menudo, después de llamar a `fork()`, el proceso hijo llama a `exec()` para cargar un nuevo programa en su espacio de memoria.