**LA BIBLIA DE LINUX**

Todas las distribuciones de Linux (Debian, Ubuntu, Kali Linux…) Provienen de un kernel llamado GNU/Linux.

Distribuciones cronológicas: <https://upload.wikimedia.org/wikipedia/commons/1/1b/Linux_Distribution_Timeline.svg>

Distribuciones principales de GNU/Linux para Ciberseguridad:

* Ciberseguridad para principiantes:
  + Kali Linux: Basado en Debian, con muchas (demasiadas) herramientas. Está bien para principiantes. Se usa para seguridad ofensiva o defensiva. Sobre todo para pentesting.
  + Kali Purple: Basado en Debian, sobre todo para seguridad defensiva (Blue team).
  + Parrot: Debian. La diferencia con Kali es el entorno del escritorio, pero se parecen bastante.
  + BlackBuntu: Basado en Ubuntu (Ubuntu se basa en Debian).
* Ciberseguridad para anonimato y privacidad:
  + Tails: Para anonimato y privacidad.
* Ciberseguridad para informática forense:
  + Sift WorkStation.
  + Caine.

## **Estructura de Ficheros de Linux:**

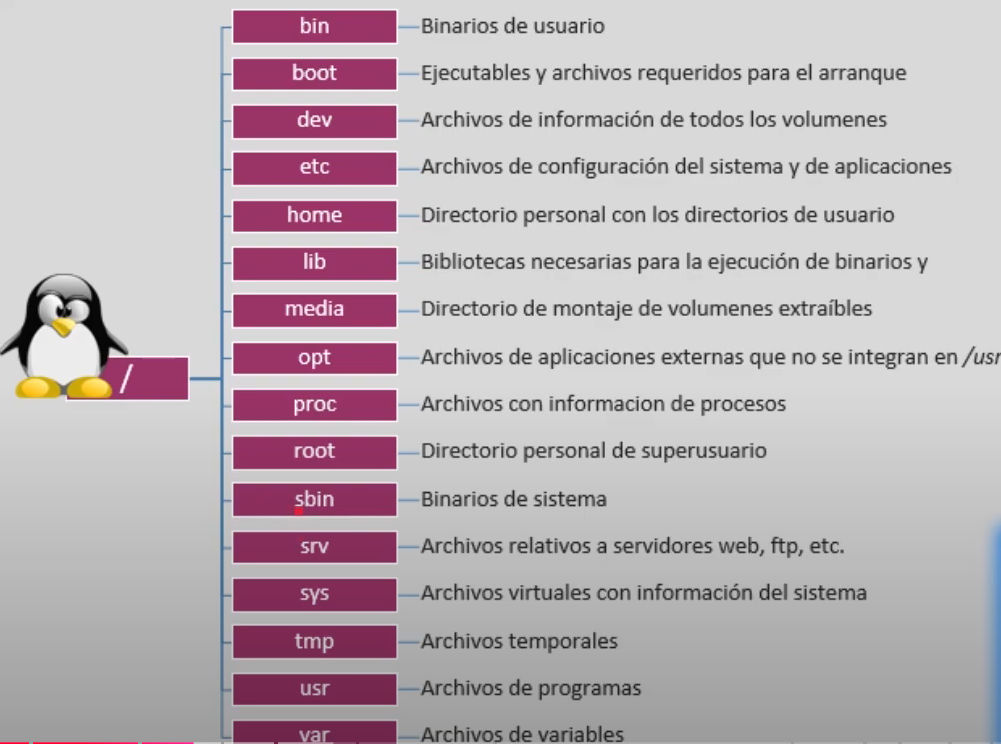
*INODOS, BLOQUES Y SUPER BLOQUES*

* **INODOS:** los Inodos contienen metadatos de todos los ficheros y directorios. Todos los ficheros y directorios van a tener asociados su Inodo y es de esta forma que el SO va a referenciarlos, mediante su Inodo, no mediante su nombre.
* **Bloques:** Es una unidad de almacenamiento donde se guardan los datos de los ficheros. Aquí relacionamos Ficheros con Inodos.
* **Super Bloques:** Contiene información crucial sobre el sistema de ficheros.

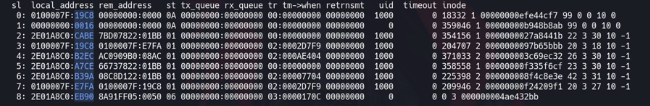
*DIRECTORIOS*

La raíz es la base de todo, esta se representa con /. De esta derivan todos los directorios.

Estos son los directorios más importantes:

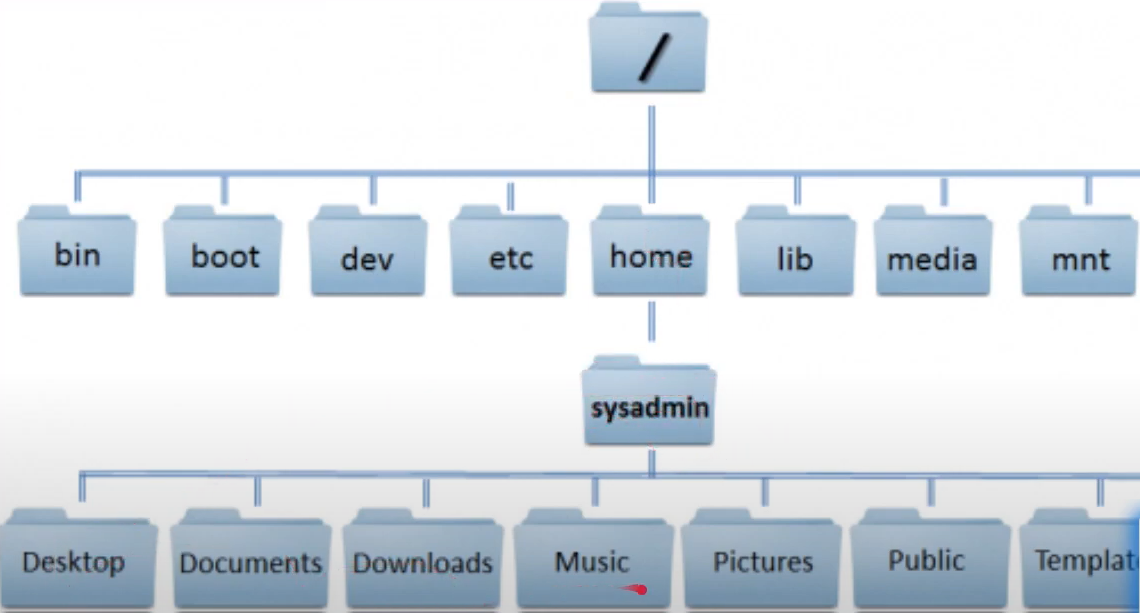


* Bin: Contiene los programas que se pueden ejecutar por TODOS los usuarios.
* Boot: Aquí se almacenan los archivos para el arranque del sistema.
  + /boot/grub: Es un gestor de arranque.
* Sbin: Ejecutables esenciales para el arranque del sistema y su recuperación.
* Dev: Archivos de información que hacen el puente entre el hardware y el software. Tanto hardware real como dispositivos virtuales.
* Etc: Archivos de configuración del sistema y aplicaciones.
  + /etc/passwd: Aquí se guardan datos de los usuarios del sistema. Nombre de usuario, Id, Id del grupo, información del usuario, Home…
  + /etc/group: Contiene información sobre los grupos del sistema.
  + /etc/shadow: Almacena las contraseñas de los usuarios de manera encriptada.
  + /etc/fstab: aquí se definen como y donde se montan los sistemas de archivos. Principalmente contiene información sobre las particiones.
  + /etc/network/interface: Aquí definimos la configuración de red para las interfaces de red. Funciona en la mayoría de distribuciones, excepto en las muy recientes.
  + /etc/hostname: Contiene el nombre del host del sistema.
  + /etc/resolv.conf: Aquí se definen los servidores DNS.
* Home: Directorio personal donde tenemos el directorio de cada usuario y de este descienden el escritorio, documentos, descargas… Cada usuario va a tener su directorio Home.
* /proc/net/tcp: En este directorio tenemos los puertos abiertos en hexadecimal.



Si pasamos a decimal la línea en azul nos dice que puertos están abiertos, cada línea es un puerto.

Adicionalmente, en este directorio vamos a tener directorios ocultos que empiezan por un punto, y dependiendo de la shell que uses vas a tener uno llamado .bashrc, zshrc…. Son importantes mencionarlos porque vamos a poder modificar o definir los alias, setear variables de entorno y realizar configuraciones permanentes.



* Media: Sirve para gestionar cuando enchufamos USB’s o cualquier hardware externo.
* Root: Es el Home del Administrador.
* Tmp: Archivos temporales que se eliminan cada vez que se apaga el equipo.
* Var: en este directorio es importante destacar que podemos ver los logs del sistema.
* Usr: Contiene los ficheros y directorios compartidos con todos los usuarios.
  + /Usr/bin/: Aquí encontramos los ejecutables.
  + /Usr/lib/: Aquí encontraremos las librerías.

Caben destacar dos directorios especiales que no se almacenan en los discos duros. Son sistemas de archivos virtuales mantenidos por el kernel del SO. Estos directorios sirven de interfaz entre el kernel y el espacio de usuario. En ellos podemos realizar configuraciones del sistema en tiempo real. Interactúas directamente con el Kernel en lugar de leer los datos del disco duro.

* /proc/: Nos muestra información sobre los procesos en ejecución, sobre el hardware del sistema e información básica del sistema.
* /sys/: Estructura de directorios que representa el Hardware del sistema. Este es bastante importante.

*ENLACES DUROS Y BLANDOS*

* **Enlace duro:**  Si por ejemplo tenemos un archivo llamado pepe.txt en dos directorios distintos. Un enlace duro es el que relaciona este archivo por su Inodo. Esto quiere decir que son el mismo archivo, puesto que tienen el mismo Inodo. Si se modifica uno, el otro también es modificado.
* **Enlace blando:** Este sería lo que en Windows es hacer una copia. Tenemos dos archivos llamados pepe.txt en dos directorios distintos y en esta ocasión si se modifica uno el otro no se modifica.

*PERMISOS*

* **Lectura (W):** Permisos de escritura.
* **Escritura (R):** Permisos de lectura.
* **Otro (o):** Engloba el resto de usuarios que no son el propietario.

*SHELL Y TIPOS*

El Shell es un intérprete de comandos, que interpreta un comando a lenguaje máquina.

* Sh: Es la Shell más básica y viene por defecto en la mayoría de distribuciones.
* Bash: Surgió de Sh, es como Sh con proteínas. Esta la vas a encontrar en la mayoría de distribuciones de forma predeterminada. También puede usarse como lenguaje de programación.
* ZsH: Recoge características de otras Shells. Es una mejora de Bash.
* Hay muchas shells, pero estas son unas de las más importantes.

En la Shell, si tenemos el símbolo $ estamos como usuario normal, si el símbolo es # estamos como administrador.

Nota: Los ejemplos los vamos a realizar con Kali Linux y en bash.

*VARIABLES DEL ENTORNO*

Lo primero, para saber que variables de entorno tenemos en este momento utilizamos el comando env o printenv:

Si queremos ver el valor que tiene una variable, por ejemplo, si tenemos una variable llamada HOME y queremos saber su valor, utilizaríamos el comando:

Si queremos definir una variable, por ejemplo, queremos crear una variable llamada pentágono con un valor de 5:

De la misma manera nos sirve para modificar el valor de una variable ya existente.

Nota: Estas variables son temporales, se borran al apagar el equipo.

Si queremos mantener el valor de la variable, pero añadiendo algo adicional:

Para eliminar una variable utilizamos:

**¿QUE VARIABLES DE ENTORNO VIENEN POR DEFECTO EN EL SISTEMA?** Vamos a repasar algunas de las más importantes:

* PATH: especifica las rutas en las que el sistema busca los ejecutables.
* HOME: Especifica la ruta home del usuario que la solicite.
* USER: Especifica el nombre del usuario actual.
* SHELL: Especifica la consola que estamos usando.
* TERM: Especifica el terminal que estamos usando.
* EDITOR: Especifica el editor de texto predeterminado.
* LANG: Especifica el idioma y localización.

**Crear una variable persistente**

Para hacerlo debemos añadirla de forma manual en el fichero de la Shell que utilicemos.

* .bashrc: Si estamos usando el bash.
* .zshrc: Si estamos usando la Shell zsh.

Dentro del documento creamos la variable de la forma en que creamos una variable temporal.

*RUTAS*

* Rutas absolutas: Las que ponemos desde la raíz, la ruta completa.
* Rutas relativas: Las que utilizamos desde una ubicación en la que ya estamos, omitiendo los directorios que hay hasta llegar a nuestra ubicación.
* .. : Si utilizamos .. en la Shell, significa que nos dirigimos al directorio padre de donde nos encontramos.

## **COMANDOS**

Dentro de los comandos hay que diferenciar entre parámetros y argumentos. Los parámetros son aquellas instrucciones que modifican un comando, por lo general vienen con un guion delante. Por ejemplo -l delante de un comando.

Por otro lado, los argumentos son datos necesarios para que un comando pueda funcionar, pueden ser obligatorios u opcionales. Por ejemplo, un /home con el comando ls, es opcional y nos listaría lo que hay en ese directorio.

Que utilicemos uno no significa que no podamos utilizar el otro, por ejemplo.

*COMANDOS DE AYUDA*

* MAN: Viene de manual y te dice que hace un determinado comando. Por ejemplo, si pongo man ls, nos indica que hace el comando ls.
  + Una vez dentro de manual podemos filtrar para encontrar más rápido lo que nos interesa. Ej, dentro del manual queremos buscar tamaño.
* INFO: Similar a man, pero algunas veces puede dar más información.
* WHATIS: Nos provee una descripción breve del comando que consultemos. Por ejemplo,
* APROPOS: Ayuda a encontrar comandos en base de una palabra clave. Por ejemplo, si queremos encontrar comandos de directorio, pondríamos , Lo cual nos listaría los comandos relacionados a esta palabra. Es lo mismo que usar el comando
* WHEREIS: Localiza el archivo binario en la página de manual de un comando. Por ejemplo, si uso , nos da el directorio del binario de ese comando.
* WHICH: Te muestra la ruta del archivo ejecutable asociado a un comando que le pases. Ej, . En este archivo están las ordenes que se van a ejecutar cuando se usa ese comando.

es decir, si por ejemplo un comando como es Whoami (el cual me indica que usuario soy), que no es más que una ruta relativa al archivo binario que contiene las instrucciones para ejecutar ese comando, yo le pongo un which delante, me va a devolver la ruta absoluta donde se encuentra ese archivo que ejecuta la instrucción Whoami (/usr/bin/whoami).

Si yo en consola pongo directamente la ruta /usr/bin/whoami me va a devolver exactamente lo mismo que si yo pongo solo whoami.

En caso de que no exista en la máquina el comando which, hay alternativas como el comando command -v whoami, que hace lo mismo.

La mayoría de comandos en Linux tienen este parámetro:

* -h o –help: ls -h. Nos dá información sobre el mismo.
* HELP: El comando help no es lo mismo que el parámetro help, con este comando podemos ver info de comandos que son únicos de la Shell BASH. No sirve en otra Shell que no sea BASH.

*SSH*

La mayoría de distros ya tienen su SSH instalado, según la distro se activa de una manera u otra, pero una vez tienes activado el ssh podemos conectarnos a nuestra máquina o usar esta para conectarnos a otras de forma remota.

Una vez tenemos descargado e instalado SSH:

* Sudo systemctl status ssh -> Nos indica si tenemos SSH activado.
* Sudo systemctl enable ssh -> Habilita que podamos activar SSH
* Sudo systemctl start ssh -> Inicia SSH en nuestra máquina.

Nota: Algunas veces el Firewall no nos deja conectarnos puesto que no permite conexiones por el puerto. Para este caso dependiendo de tu firewall tienes que habilitar el puerto.

* Ssh usuario@IP -> Con este comando nos conectamos a una máquina remota.
  + Ssh usuario@enlace -> En ocasiones podemos usar un enlace de una web para conectarnos.
  + -p -> Nos permite indicar por que puerto nos queremos conectar. Por defecto nos conecta por el 22.
* Hay ocasiones en las que se ha modificado la máquina para que al intentar conectarnos nos expulse. Una posible solución es enviar algún comando junto a la conexión ssh para intentar entrar. Por ejemplo, si enviamos un bash, puede ser que nos abra una bash antes de expulsarnos y nos deje quedarnos.
* Sshpass -p ‘contraseña’ Si ponemos esto delante del comando ssh podemos poner la contraseña directamente y no nos va a pedir la contraseña al intentar entrar al usuario.

No siempre nos vamos a tener que conectar por SSH con una contraseña, existen las claves públicas y privadas, estas se encuentran o se deben crear en el directorio /home/Tuusuario/.ssh.

* Ssh-keygen -> Este comando nos crea un par de claves. Una pública y una privada. Nos permite crear una passphrase.
* Autorized\_keys -> Este es un archivo que tiene que estar en el directorio /home/Tuusuario/.ssh. En este archivo añadiremos las claves públicas de los equipos que queramos que se conecten a nuestra máquina sin necesidad de pedir ninguna credencial. Para hacer la conexión desde el otro equipo, simplemente habría que hacer una conexión ssh normal, pero no te va a pedir contraseña. Si el archivo no existe lo podemos crear.

Otra forma de conectarse es autorizando la clave privada y que todo aquel que tenga esa clave se puede conectar a mi equipo. Es necesario añadirle el privilegio 600 a las claves para poder usarlas.

*COMANDOS DE NAVEGACIÓN Y GESTIÓN DE FICHEROS Y DIRECTORIOS*

* CD: Change directory. Cambiamos de directorio, admite tanto rutas relativas como absolutas. Por defecto te envía a Home.
  + ~ -> Lo podemos añadir a cd. La Virgulilla representa el directorio Home.
* PWD: Imprime por pantalla el directorio donde te encuentras.
* LS: List. Lista los ficheros y directorios que hay dentro de un directorio. Cuenta con varios parámetros, los más usados son:
  + -la -> Muestra ficheros/directorios ocultos.
  + -l -> lista en formato largo, viendo así más detalles.
  + ../ -> hace referencia al directorio anterior.
  + ./ hace referencia al directorio actual.
* TOUCH: Crea ficheros vacíos.

Tiene algunos parámetros como:

* + -a -> Modifica la hora de acceso al fichero, se usa en caso de que quieras cambiar la hora en la que accediste a un fichero.
  + -m -> Cambia la fecha de modificación.
* CP: Copy. Copia un fichero o directorio. Admite rutas absolutas o relativas.
  + -i -> Fuerza a pedir confirmación en caso de que haya un archivo con el mismo nombre, para no machacarlo.
  + -r -> Sirve para copiar directorios enteros, con todo lo que el contenga.
* MV: Move. Mueve ficheros de un sitio a otro. También sirve para cambiar el nombre de un fichero.
  + -i -> Pide confirmación en caso de que haya un archivo con el mismo nombre.
  + -u -> Sobrescribe el archivo si el archivo a sobrescribir es mas antiguo.
* RM: Remove. Elimina ficheros.
  + -i -> Pide confirmación antes de eliminar.
  + -r -> Se usa principalmente en directorios, cuando este tiene muchos otros directorios. Para borrarlo todo usamos este parámetro.
* MKDIR: Make Directory. Creas un Directorio.
  + -p -> Permite crear sucesiones de directorios, uno dentro de otro.
* RMDIR: Remove Directory. Elimina directorios.
  + –ignote-fail-on-non-empty -> Ignora los fallos cuando eliminamos un directorio que no está vacío.
* CAT: se usa para abrir un fichero. Pero realmente este comando es Concatenate y su origen fue para concatenar archivos. Ejemplo de abrir fichero:
* Mktemp-> Nos permite crear un directorio o archivo de trabajo temporal. En el podremos empezar a crear archivos en máquinas donde no tenemos permisos.

Ejemplo concatenar:

+1+2)

* + -n -> Cuenta el número de líneas.
  + Hay ocasiones en las que un archivo tiene nombres extraños que al abrirlos de la forma tradicional con CAT no funciona.
    - - -> En caso de que haya un guion será necesario abrirlo pasando la ruta entera.
    - Archivos con espacios -> Realizamos un CAT y ponemos el nombre entre dobles comillas.
* HEAD: Muestra las primeras 10 filas del documento.
  + -nx -> muestra las líneas que le indiquemos de un fichero.
* TAIL: Lo contrario a HEAD. Lee las últimas 10 líneas. De la misma forma tiene el parámetro -n.
  + -f -> Con este parámetro seguimos el contenido de un fichero en tiempo real. Esto es útil para ir viendo los logs, que se van almacenando en las últimas líneas.
* LESS: Sirve para ver documentos de forma paginada, es decir, verlos por páginas. Muy útil para ficheros grandes.
  + -f -> Vemos el fichero en tiempo real.
  + -n -> Visualiza el número de líneas.
  + +g -> Muestra el final del fichero.

*Concepto stdin, stdout y stderr*

STD vienen de estándar y en Linux en comandos tenemos 3 estándar.

* STDIN: Standar IN: Se relaciona al número 0 y son todos los datos que son recibidos por un programa o comando de un usuario. Por ejemplo, lo que escribimos en el Shell para introducir un comando.
* SDTOUT: Standar OUT: Se relaciona al número 1 y es aquello que devuelve un programa o comando cuando se ejecuta correctamente al usuario o a otro programa. Por ejemplo, lo que nos devuelve el comando ls al ejecutarse correctamente.
* STDERR: Standar ERROR: Se relaciona al número 2 y es todo aquello que nos devuelve un programa o comando cuando no es ejecutado correctamente.

Los números que se relacionan con los estándares se utilizan para las redirecciones, es decir, para guardar lo que nos devolvería en un archivo. Se hace con STDOUT y STDERR Por ejemplo:

Con este comando guardaremos la lista que nos devolvería ls en archivo.txt, redirigiendo la información al archivo.

Ejemplo 2:

Esto redirigiría los mensajes de error al archivo.txt.

Hay un archivo /dev/null donde se suele enviar todo a modo de agujero negro. Si queremos enviar tanto STDERR como STDOUT utilizamos &

*Descriptores de archivo*

Además del 0, 1 y 2, también tenemos otros números que nos sirven para añadir salidas a archivos concretos. En este caso si por ejemplo yo me creo un archivo al que le agrego la propiedad del número 3.

Ahora yo puedo enviar las salidas a este archivo el cual es de lectura (<) y escritura (>) definidos con el menor y mayor que. Por ejemplo.

También me sirven el 4, 5, 6… No sé exactamente cuántos hay asignados para este propósito.

Para quitar la propiedad de descriptor utilizamos

De esta forma quitamos a dicho archivo la propiedad de descriptor, PERO el archivo ni se elimina ni pierde su contenido.

**Nota:** Se puede asignar mas de un número de descriptor a un mismo archivo.

*Comandos de gestión de usuarios*

* USERADD y USERDEL: Añade y elimina usuarios, pero no están optimizados.
* ADDUSER: Crea usuarios en el directorio /etc/passwd. Y meteremos datos para rellenar la ficha del usuario. Se realiza con el comando sudo.
* DELUSER: Elimina un usuario del SO.

Con el parámetro --remove-home eliminamos también el directorio home de ese usuario.

* USERMOD: Nos permite modificar las propiedades de un usuario. Tiene muchos parámetros y es un comando muy potente. Dos de los más importantes.
  + -l -> Permite cambiar el nombre de login del usuario.
  + -d -> Permite cambiar el directorio HOME del usuario.
  + –aG -> Nos permite añadir un usuario a un grupo.

* PASSWD: Cambia la contraseña de un usuario. Los parámetros más relevantes son:
  + -l -> Bloquea la contraseña del usuario, es como desactivar la cuenta de ese usuario.
  + -u -> Desbloquea la contraseña del usuario, es decir, reactiva la cuenta.
* ID: Nos identifica la información del usuario actual o del que le indiquemos.
  + -u -> Nos muestra el UserId (UID) del usuario.
  + -g -> Nos muestra el GroupId (GID) del usuario
* WHO: Nos permite saber quien está conectado al sistema.
  + -u -> Nos muestra una lista de los usuarios conectados y la hora de su conexión.
* SU: Switch User. Sirve para cambiar de usuario. Si no se especifica nombre de usuario se cambiará al usuario Root.
  + -l o --login -> Nos muestra un entorno similar al que tendría el usuario si se conectara.
  + -c -> Permite ejecutar un comando con los permisos del usuario que le indiquemos, sin tener que loguearnos con ese usuario.
* SUDO: Nos permite ejecutar comandos con permisos de otro usuario, es similar al comando SU con el parámetro -c. Si no ponemos usuario por defecto coge el usuario Root.
  + -u -> Especificamos el usuario con el que ejecutaremos el comando.
  + -i -> Proporciona entorno similar al que tendría el usuario si se conectara.

Este comando es más potente que SU.

* WHOAMI: Nos indica con que usuario estamos logueados.
* FINGER: Proporciona información sobre un usuario en particular, como cuando inicio sesión, etc…

*Comandos de gestión de usuarios*

* GROUPADD GROUPDEL: Comandos poco optimizados para la creación y eliminación de grupos.
* ADDGROUP: Se usa con SUDO y sirve para crear grupos en el SO.
* DELGROUP: Lo mismo que con ADDGROUP, pero para eliminar.
* GROUPMOD: Sirve para modificar parámetros de grupos. Es un comando poco optimizado.
  + –n -> Sirve para cambiar el nombre del grupo.
* GPASSWD: Su función es administrar las contraseñas de los grupos.
  + –a -> Nos sirve para añadir un usuario a un grupo.
  + –d -> Eliminar usuario de grupo.
* GROUPS: Consulta los grupos a los que pertenece el usuario en cuestión. Si no se pone nada por defecto te consulta tu usuario.

*Comandos de gestión de paquetes*

PAQUETE: En Linux un paquete es una colección de archivos la cual permite la instalación de un programa y da toda la configuración que se tiene que hacer en un principio para que esta funcione.

REPOSITORIO: Almacenamiento centralizado donde se mantienen paquetes de software. De estos descargamos los paquetes para instalar las aplicaciones que necesitemos.

DEPENDENCIAS: Es software adicional que se necesita para que un programa funcione correctamente.

* APT: Advance Packaging Tool. Sirve para descargar programas con todas sus dependencias. Viene por defecto en distribuciones basadas en Debian. Pero hay distribuciones donde tendrías que descárgalo.
  + Update: Actualiza la lista de paquetes disponibles, así como sus versiones.
  + Upgrade: Actualiza a la versión más reciente los programas que ya tenemos instalados.
    - -y -> Con este parámetro le decimos que no nos pida confirmaciones a la hora de descargar o actualizar un paquete.
  + Full-upgrade: Es como una versión con esteroides de Upgrade. Maneja la instalación de las dependencias de forma inteligente.
  + Install: Nos sirve para instalar paquetes.
  + Remove: Eliminamos paquetes.
  + Auto-remove: Sirve para desinstalar de forma automática dependencias que fueron agregadas al instalar paquetes antiguos y ya no se usan.

Nota: lo normal al principio de cada instalación del SO es hacer un update y luego un upgrade, pero esto en ocasiones puede causar que deje de funcionar correctamente el SO, puesto que hay muchos programas y puede haber incompatibilidades.

* DPKG: Debian Package. Es una versión un poco mas pro de APT. Sirve para hacer instalaciones mas manuales de programas .deb haciéndolo a más bajo nivel que APT.

Parámetros:

* + –i -> Install.
  + –l -> Nos lista los paquetes instalados en el sistema.
  + –r -> Remove. Elimina un paquete de forma parcial, mantiene los archivos de configuración en el sistema.
  + –p -> Eliminación por completo de un paquete.
* APT-GET: Versión antigua y un poco más estable de APT. Adecuada cuando se usa dentro de un Script.
  + Dist-upgrade -> parámetro que además de hacer un upgrade, también es capaz de manejar de forma inteligente los cambios en las dependencias con nuevas versiones a nivel de paquetes.

-y

* APT-CACHE: Realiza búsquedas y obtiene información sobre los paquetes almacenados en la cache.
  + Depends -> Nos muestra las dependencias de un paquete.
  + Policy -> Nos da la política instalada de un paquete, incluyendo versiones disponibles y de que repositorio se va a instalar.

Por último, para esta sección vamos a ver algunas utilidades de DPKG que son bastante usadas.

* DPKG-QUERY: Se utiliza para hacer consultas a la base de datos de DPKG.
* DPKG-RECONFIGURE: Se usa bastante si se usan máquinas virtuales. Sirve para reconfigurar un paquete después de que haya sido instalado.
* DPKG-TRIGGER: Permite activar eventos en el sistema, esto es activadores o disparadores que se ejecuten cuando ocurra alguna acción.

Es vital mencionar el fichero: /etc/apt/source.list.

De este fichero va a sacar APT las rutas de los repositorios

Y el fichero: /etc/apt/source.list.d/.

En este fichero hay más rutas para otros repositorios. Es como un fichero menos importante que el anterior y el cual se suele modificar antes que modificar el principal que sería source.list.

*Gestión de redes*

**Nomenclaturas** usadas para las interfaces de red en LINUX, hay dos tipos de nombres tradicionales y predictivos:

**Tradicionales**: No son persistentes, esto es que pueden cambiar cuando se reinicia el SO. Como Eth0, eth1 o wlan0, wlan1.

**Predictivos:** Son persistentes. Se basan en la ubicación física o tipología de hardware. Son los que probablemente te llegues a encontrar. Como, por ejemplo:

* En: Para interfaces de Ethernet.
* Wl: Interfaces de Wlan (red de área local inalámbrica).
* WW: Para interfaces Wwan (redes inalámbricas de área amplia).
* P(nº bus): El nº bus indica que número de bus.
* S(nª dispsitivo): El nº dispositivo indica que ranura del bus ocupa.

**Ejemplos:**

* Interfaz Enp2s0 -> en se refiere a Ethernet, p2 nos indica que estamos en el bus PCI 2 y s0 que dentro del bus 2, estamos en la primera ranura (dispositivo 0)

**COMANDOS LINUX PARA REDES**

**Ifconfig:** Lista todas las interfaces de red activas, mostrando detalles como IP, máscara… Se usa normalmente como consulta, pero en realidad se está hecho para configurar las interfaces de red.

* Ifconfig eth0 -> te muestra solo esa interfaz de red.
* Ifconfig eth0 up/down -> activas o desactivas esa interfaz de red.
* Ifconfig eth0 192.168.11.33 -> pones una IP a una interfaz de red.

Si ifconfig no viene instalada, se puede instalar con el comando apt install net-tools.

**IP:** Mostrar y manipular rutas, dispositivos, etc…

**IP route(ip ro):** Muestra la tabla de ruta del kernel. Se pueden añadir nuevas rutas personalizadas.

* Ip ro add 192.168.2.0/24 via 10.0.2.4 -> Para decirle por donde quiero enviar.
* Ip route get 8.8.8.8 -> Nos dirá por qué puerta de enlace iremos a la dirección 8.8.8.8

**netstats:** Muestra tablas de enrutamientos, miembros multicast, etc…

* Netstat -t -> Muestra conexiones por TCP.
* -u -> UDP
* -l -> Muestra procesos en escucha
* -p -> Muestra el PID
* -e -> Muestra información extendida

**ping:** Herramienta que permite ver la conectividad con otro dispositivo.

* Ping marca.com -> se puede enviar a dominios.
* Ping -c11 -> Enviaríamos 11 paquetes.

**Tracerout:** Muestra la ruta que toman los paquetes hasta llegar a su destino.

* -m 33 -> Permite especificar el máximo número de saltos hasta llegar a su destino, en este caso serían 33 saltos.

**Nslookup:** Sirve para hacer consultas en los servidores DNS.

* -type=MX -> especificas el tipo de registro que quieres ejecutar. En este caso MX

**Hostname:** Muestra o configura el nombre del SO.

**Lsof –i:Puerto:** Nos indica que servicio está corriendo en un puerto.

**NC:** Comando netcat, herramienta sencilla que sirve para manipular puertos.

* + Abrir puerto de escucha temporal. De forma que podamos conectarnos a ese puerto y enviar cosas.

**Ncat**: Comando como el anterior pero que tienen distintas herramientas, por ejemplo, este comando tiene una herramienta para encriptación SSL que nc no tiene.

Para mirar si un puerto está activo se le envía una cadena vacía (Se hace en bash, puesto que en zsh no deja)

*Comandos de gestión de archivos comprimidos*

* **TAR:** Herramienta para crear archivos comprimidos. Utiliza la extensión .tar.
  + **–**c -> Indica la creación de un fichero.
  + **–**z -> Utiliza GZIP para la compresión/descompresión.
  + **–**v -> Muestra ficheros procesados.
  + **–**f -> Especifica el fichero de salida.
  + **–**x -> indica extracción de ficheros.

**Un ejemplo de comprimir** un archivo llamado Documentos\_Comprimidos en una determinada ruta.

**Un ejemplo de descomprimir**

* **GZIP:** Herramienta simple para comprimir ficheros individuales. Tienen extensión .gz
  + **–**d -> Indica descomprimir archivos.
* **ZIP**: Lo mismo que el anterior, pero les añade la extensión .zip. En este caso también le decimos el nombre del fichero que vamos a generar.
  + **UNZIP**: Para descomprimir.

**Nota**: La idea es utilizar cada comando según la extensión que tenga el archivo a descomprimir o la que queramos poner al comprimirlo. Pero puede haber ocasiones en que nos encontremos archivos con mas de una extensión, por ejemplo un fichero .tar.gz. Informarse de como descomprimirlo según cada caso. Si tomamos este ejemplo se descomprimiría con TAR.

-**7z:** Descompresor de archivos muy completo y recomendado para usar.

* L -> Detecta que hay dentro, cual será el archivo que va a descomprimir.
* X -> con este parámetro te va extraer el archivo independientemente de si está en tar, gzip, etc…

*Operadores lógicos*

* **&&**: Actúa como un AND. Ej. (comando) Y (comando). Para que se ejecute el segundo comando, el primero tiene que haberse ejecutado con éxito.
* **||**: Equivale a un OR. Ej. (comando) o (comando). Se utiliza para ejecutar un segundo comando solamente si el primero ha fallado.
* **!**: Equivale a un NOT. Invierte el resultado del comando. Ej. !(comando). Si el comando no es exitoso no te enviará mensaje de error y viceversa.
* **;**: Similar al OR, pero con más libertad ya que el siguiente comando se ejecutará independientemente de si el anterior se ejecutó con éxito o no. Garantizando que todos los comandos se ejecuten.

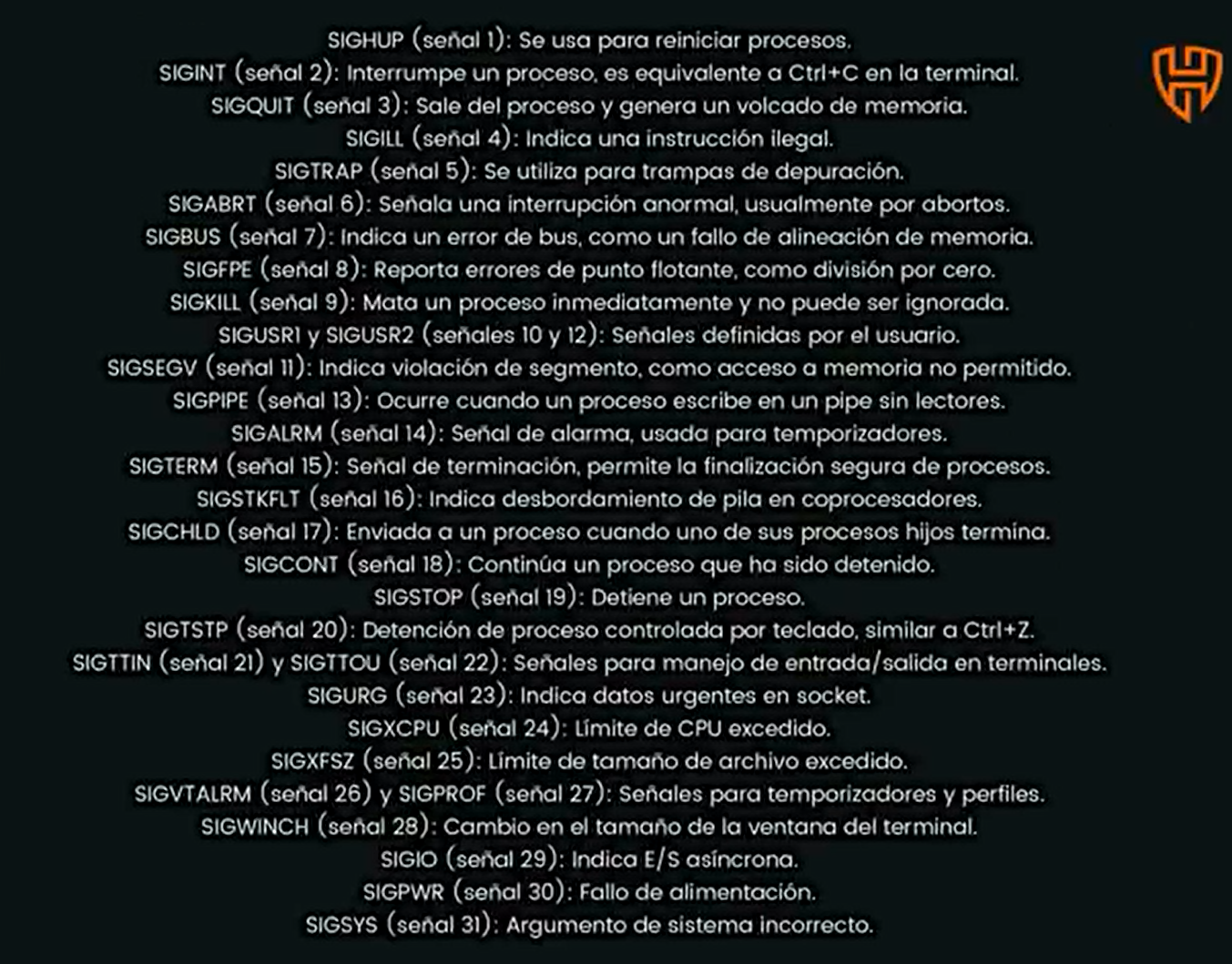
*Comandos de gestión de procesos*

**Proceso:** Un proceso es una instancia de una aplicación que está en ejecución. Una aplicación tiene uno o varios procesos que se ejecutan para que esta se pueda ejecutar correctamente. Cada proceso en Linux consta de varios puntos clave.

* PID: Process IDentifier. Único para cada proceso.
* Tipos de procesos: Hay dos tipos de procesos.
  + Procesos de usuario: Procesos iniciados y/o controlados por los usuarios. Ej. Editor de texto o cualquier comando que ejecutemos.
  + Procesos del sistema: Son transparentes al usuario. Están manejados por el SO y son necesarios para el funcionamiento de este.
* Estado de los procesos: Los procesos pueden tener un estado asignado:
  + Running: Proceso corriendo.
  + Sleeping: Proceso dormido.
  + Stoped: Proceso detenido.

Explicado esto veremos los comandos asociados a la gestión de procesos.

* **PS**: Muestra información de los procesos activos. Consta de algunos parámetros:
  + **a :** Muestra los procesos de todos los usuarios del sistema(all).
  + **u :** Muestra procesos del usuario (user).
  + **x :** Muestra procesos sin tty asociada.
* **TOP**: Muestra en tiempo real los procesos en ejecución, así como su consumo de recursos.
* **KILL**: Mata un proceso. La sintaxis sería. Kill (señal) (PID). Dependiendo de la señal realizamos una acción u otra.



* **KILLALL**: Mata todos los procesos asociados con el nombre que le demos. Ej. Si pongo killall nano, mata todos los nanos que estén abiertos.
* **&:** Poner & al final de un comando nos permite ejecutarlo en segundo plano.
* **JOBS:** Con este comando podemos listar los procesos en segundo plano. Cada proceso que nos liste tendrá un identificador numérico para diferenciarlos.
  + Si en algún momento queremos traer un proceso en segundo plano al primer plano usaremos el comando fg y el identificador de ese proceso.

*Redirecciones*

* **>** -> Redirige por defecto el stdout donde le indiquemos. Este comando sobrescribe el fichero.
* **>>** -> Lo mismo que el anterior pero no sobrescribe el fichero.
* **|** -> Tuberías o PIPS. Permite encadenar comandos, pasando el stdout o stderr que vamos obteniendo como stdin en el comando siguiente en forma de argumento. Ej. Realizamos un listar y el resultado se lo pasamos al comando grep (Realiza búsquedas. comando que veremos más adelante)

Podemos añadir tantas tuberías como queramos.

* **XARGS:** Hay veces que, tras usar una tubería, el comando que recibe el stdout del anterior comando como argumento (stdin) no puede procesarlo correctamente porque son muchos argumentos y este sólo puede procesarlos de uno en uno. Para estos casos usamos este comando.

Listamos todos los archivos terminados en .txt y por último los eliminamos.

*Comandos de búsqueda*

* **FIND:** Sirve para hacer una búsqueda. Ej. Queremos buscar los ficheros .txt en /home/. La sintaxis sería find (directorio en el que buscar) (por qué campo vamos a buscar) (dentro de ese campo que filtro usamos para hacer la criba).
  + - En este caso -name se refiere a que vamos a usar el campo nombre de los ficheros
    - En este caso \* (significa cualquiera) que es cualquier fichero que termine en .txt
    - -group -> nos busca por grupo
    - -type f o d -> Nos indica si queremos filtrar por directorios o por ficheros.
    - -writable -> Nos sirve para filtrar por ficheros que tienen capacidad de escritura.
    - -executabel-> Lo mismo que el anterior, pero para el permiso de ejecutar.
    - . -> Si ponemos find . nos lista todos los archivos de un directorio.
* **GREP:** Buena herramienta que filtra patrones dentro de ficheros ya que utiliza expresiones regulares, permitiendo hacer búsquedas más complejas. Es muy útil con tuberías.
  + **–**i -> Elimina el keySensitive.
  + **–**c -> Cuenta el número de líneas que nos da el patrón de búsqueda.
  + **-**v -> va seguido de una cadena y sirve para descartar los resultados que contentan dicha cadena. Si añadimos una E podremos descartar mas de una cadena.
* **LOCATE:** Sirve para localizar ficheros dentro del sistema. Usa su propia base de datos preconstruida (esta se actualiza por lo general 1 vez al día). Ej. Si queremos localizar un archivo concreto.

Esto nos lista TODOS los archivos en el sistema que contienen esta palabra.

**Nota:** Podemos actualizar la base de datos de Locate manualmente con el comando.

*Editores de ficheros desde la terminal*

Básicamente utilizaremos dos editores en Linux, VIM (versión mejorada de un editor llamado VI) y NANO.

* **VIM:** Editor medio avanzado. Cuando utilizamos VIM si el fichero no existe lo crea.

Tiene varios modos:

* + **Normal:** Modo por defecto al abrir un fichero. No podemos insertar texto.
    - **H:** Con esta tecla nos movemos hacia la izquierda.
    - **J:** Hacia abajo.
    - **K:** Arriba.
    - **L:** Derecha.
    - **W:** Saltamos al inicio de la siguziiente palabra.
    - **E:** Saltamos al final de la palabra actual.
    - **Dd:** Eliminamos una línea.
    - **Yy:** Copiar una línea.
    - **P:** Pegar línea copiada.
    - **I:** Ingresamos al modo inserción.
    - **:**: Ingresamos al modo comando.
  + **Inserción:** En este si podemos añadir texto y accedemos a este presionando la letra i en el modo normal.
  + **Comando:** Nos permite realizar mediante comandos operaciones complejas.
    - **:**W 🡪 Guardar.
    - **:**Q 🡪 Salir.
    - **:**Q! 🡪 Forzar salida.
    - **:**WQ 🡪 Guardar y salir.
    - **:**/ 🡪 Buscar.

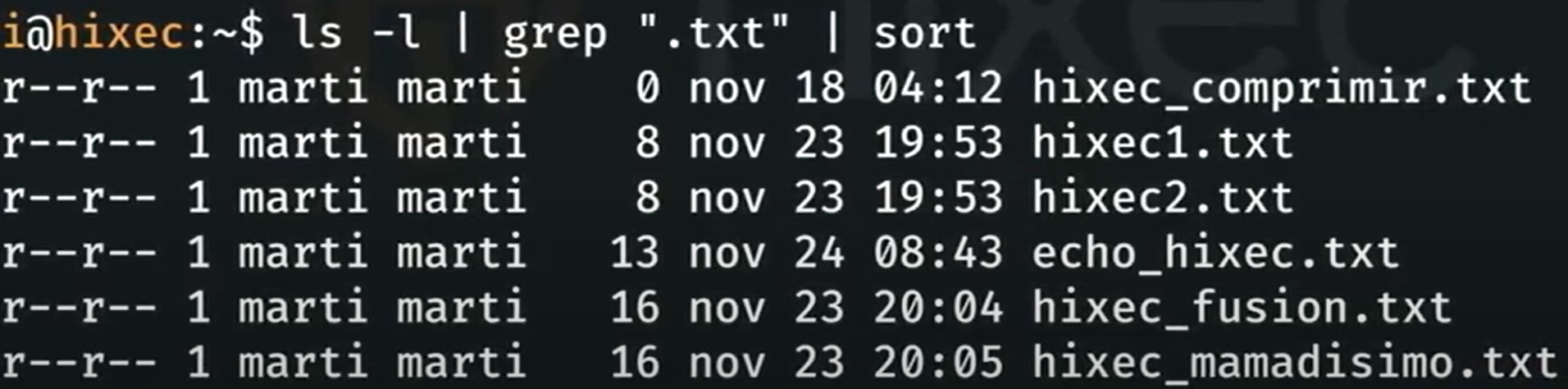
Estos sólo son los comandos más básicos, el mundo de VIM es amplio y requiere de un curso entero.

Nota: Si estamos en el modo inserción no podemos entrar al modo comando, para eso habría que salir del modo inserción mediante la tecla escape y luego si entrar al modo comando.

* **NANO:** Editor para más principiantes. Las instrucciones de uso te las da el mismo editor al usarlo. Además, tendríamos algunos parámetros importantes que podemos usar:
  + **–**m -> Habilita el uso del ratón dentro de NANO.
  + **–**l -> Muestra número de líneas.
  + **–**v -> Abre el fichero solo en lectura.
  + +X -> Siendo X un número entero que va a indicar en que línea estará el cursor al abrir el archivo.

*Comandos Adicionales*

* **SORT:** Comando para ordenar líneas de texto en ficheros. Puede ordenar alfabéticamente, numéricamente o por columnas, en este último caso si no le indicamos nada va leyendo las columnas de izquierda a derecha y en la primera columna donde detecte que hay diferencias, ordena por esa columna. Ej. En el caso de un ls la primera columna es la de permisos, si dentro de esta columna están todos los datos iguales pasa a la segunda columna que es la de usuarios, si los usuarios son todos iguales pasa a la siguiente columna y así sucesivamente hasta que encuentra una columna donde hay datos diferentes y usa esa columna para ordenar.



En este caso ordenaría por peso del archivo, pues de izquierda a derecha es el primer campo que tiene resultados distintos.

* + –k -> Especifica el número de columna por la que queremos filtrar.

Para el caso de la imagen anterior, la columna nueve sería los nombres del fichero.

* + –f -> Elimina el key Sensitive.
  + –o -> Guarda el resultado en un fichero.
  + –n -> Ordena de forma numérica. Por defecto lo hace de manera alfabética.
  + –t -> Sirve para ordenar por un campo específico.
  + –c -> verifica si el fichero ya está ordenado.
* **Uniq:** Sirve para encontrar cadenas de texto duplicadas en un mismo archivo. Para que lo considere duplicado debe de estar en líneas consecutivas, es por eso que normalmente se usa con el comando sort que lo que hace es ordenar y agrupar.
  + **-**i -> Con i desactivamos el keySensitive.
  + **-**c -> Nos muestra el número de veces que se repite cada valor.
  + **-**d -> Nos muestra solo los valores que están duplicados.
  + **-**u -> Nos muestra solo los valores que NO se repiten.
* **NC:** Netcat es usada principalmente para abrir y cerrar puertos.
  + **–l ->** Listening para escuchar por un puerto.
  + **–p ->** nos permite indicarle un puerto.
  + **–v ->** Activa modo Verbose para mostrarnos más detalles.
  + **–n ->** Evita que se realice la resolución DNS.
  + **–u ->** Usar UDP en lugar de TCP.
* **ECHO:** Nos permite imprimir por pantalla.
  + Podemos imprimir texto.
  + Una ruta.
  + –n -> Evita que se imprima una línea nueva al final del texto.
  + –e -> Habilita la interpretación de secuencias de escape. Como, por ejemplo.
    - \n : para hacer saltos de línea.
    - \t : Tabulación.
* **WGET:** Herramienta que nos permite descargar ficheros de internet.
  + **–c ->** Continuamos una descarga.
  + --no-check-certificate -> Ignora la certificación del certificado DLS.
* **CURL:** Herramienta para transferir datos desde o hacia un servidor.
  + **–x ->** Especificamos el método de solicitud HTTP.
  + **–h ->** Añadimos encabezados a la solicitud.
* **DU:** Indica el espacio utilizado por fichero o directorios.
  + **–a** -> Con esto nos muestra el uso de espacio de todos los ficheros.
* **FREE:** Muestra el estado de la memoria del sistema. Incluyendo memoria física y swap.
  + **–b ->** Nos indica los resultados en Bytes.
  + **FILE:** El comando file nos indica el tipo de archivo.
  + **STRING:** Imprime los caracteres que si son legibles en texto de un archivo.
  + **BASE64:** Nos convierte un archivo a based64
    - **-d ->** Nos convierte un archivo de based64 a texto normal.
  + **SPONGE:** Cuando queremos manipular un archivo y además guardar el resultado en ese mismo archivo, es muy típico que nos de problemas del tipo que se nos borra el archivo. Para estos casos usamos SPONGE. Sobrescribe un archivo y le mete lo que indiques. Se suele usar con el estándar STDIN, STDOUT y STDER.
  + **XXD:** Nos pone un archivo en hexadecimal
    - **-r ->** De hexadecimal nos lo pone en texto legible.
    - **-ps ->** Nos aísla solo la parte hexadecimal y emite el resto.
  + **DIFF:** Nos dice la diferencia entre dos archivos.

Nos dice en < las líneas que no están en el segundo archivo y en > las líneas que se han añadido.

* + **UNAME -r:** Muestra la versión de Kernel que tienes.
    - **-n:** Muestra la máquina en la que estamos.
  + **LN:** Creamos un enlace.
  + **TIME:** Comando para saber cuánto tiempo tarda un comando en ejecutarse.

time ls

* + **SED:** Herramienta que permite procesar y modificar texto de forma eficiente. Se usa principalmente para buscar, reemplazar, insertar y eliminar líneas de texto o cadenas de texto.
    - **Remplazar texto:**

El último comando nos daría una salida de Hola Linux.

Nota: Solo remplaza la primera coincidencia en cada línea. Para reemplazar todas usamos g:

* + - **Eliminar líneas:**

Eliminar la 3 línea del archivo.

Eliminar todas las líneas que contengan “error”.

* + - **Insertar texto:**

Insertamos texto antes de una línea con i:

Insertamos texto después de una línea con a:

* + - **Mostrar ciertas líneas:**

Mostrar solo la línea 5:

Mostrar de la línea 5 a la 10:

* + - **Concatenar comandos:**

Se pueden ejecutar varios comandos a la vez con –e:

O con “;” dentro de las comillas.

* + - **Expresiones regulares:** Con sed podemos usar expresiones regulares para realizar determinadas tareas. Pero eso es algo que no veremos en este momento.
  + **dsad**

*Atajos*

* **Atajos a nivel de línea de comandos:**
  + **Ctrl + a ->** Te mueves al inicio de la línea.
  + **Ctrl + e ->** Te mueves al final de la línea.
  + **Ctrl +b ->** Te Mueves un carácter atrás.
  + **Ctrl + f ->** Te mueves un carácter adelante.
  + **Alt + b ->** Mueve el cursor una palabra hacia atrás.
  + **Alt + f ->** Mueve el cursor una palabra hacia Adelante.
  + **Ctrl + k** -> Eliminamos lo que haya delante del cursor en la línea que estemos escribiendo.
  + **Ctrl + xx ->** Selecciona desde el inicio de la línea a la posición actual del cursor.
* **Atajos a nivel de procesos:**
  + **Ctrl + c ->** terminamos el proceso actual.
  + **Ctrl + z ->** Suspende el proceso actual. Para traerlo de vuelta utilizamos el comando **fg**.
  + **Ctrl + d ->** sale de la terminal.
* **Atajos de historiales de comandos:**
  + **Ctrl + p ->** Sirve para navegar al comando anterior del historial.
  + **Ctrl + n ->** Sirve para navegar al comando siguiente del historial.
  + **Ctrl + r ->** Nos permite buscar dentro del historial.
  + **Ctrl + o ->** complemento al anterior, si buscamos un comando con Ctrl + r, una vez lo encontremos en el terminal, con Ctrl + o se puede ejecutar.
  + **Alt + . ->** inserta el último argumento del comando anterior. También puede ser Alt + \_.
* **Atajos a nivel del emulador del terminal:**
  + **Ctrl + l ->** Limpiamos la pantalla. En realidad, no la limpia si no que echa para abajo el terminal, dejando los comandos por encima de nuestra visualización.
  + **Ctrl + s ->** hay comandos que nos dan salidas continuas muy largas que si no se detienen es imposible de leer, pues con este comando lo congelamos para su lectura.
  + **Ctrl + q ->** Con este reanudamos el desplazamiento. Complementa al anterior.
  + **Ctrl + alt + (F1-F6) ->** Nos permite cambiar entre las terminales TTI.
* **JOBS:** Nos permite ver los procesos suspendidos o en segundo plano.
* **Disown:** elimina toda la lista de trabajos lo que nos permite separarnos de un proceso creado desde nuestra terminal, haciendo que sea independiente. Ej. Si yo desde la terminal ejecuto Firefox, una vez yo cierre la terminal, Firefox se cerrará. Pero si yo lo envío a segundo plano con el comando & y luego independizo el proceso con Disown. Aun cerrando la terminal, Firefox no se cerrará, pues ahora es independiente.

*+ Atajos*

* **Flechas Arriba y abajo:**  Nos permite movernos entre los comandos según el orden en que los hemos ido usando.
* **History:** Almacena un historial de los comandos ejecutados. Estos comandos se guardan en el fichero .bash\_history. Este fichero lo tiene cada usuario.
* **Tabulación:** Completa el comando que estamos escribiendo.
* !$ 🡪 Sirve para recuperar el último argumento del comando anterior.
* !! 🡪 Sirve para recuperar exactamente el último comando ejecutado.

*Alias*

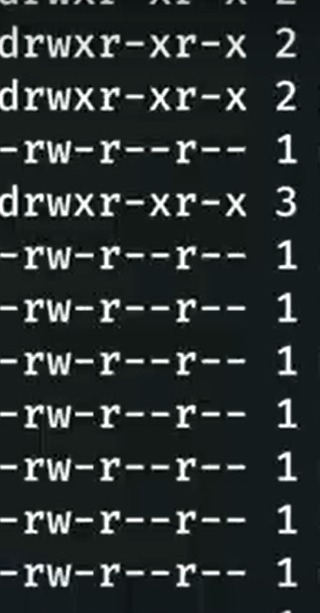
Un alias es una palabra a la que le asignaremos un comando. Para que cada vez que usemos ese alias se ejecute dicho comando. Estos son temporales. Para hacerlos permanentes es necesarios añadirlos al fichero .bashrc.

* **ALIAS:** Por defecto nos indica los alias que tenemos creados.
* **UNALIAS:** Elimina el alias.

*Permisos*

* **Chmod**: Con este comando vamos a realizar la asignación de permisos.

En Linux los permisos principales son lectura (r), escritura (w) y ejecución (x). Adicionalmente tenemos SUID, SGID, Sticky Bits y sudoers.



La primera letra que puede ser una d (directory) o un guion (fichero) nos indica si el archivo es un directorio o un fichero.

Después de este primer carácter dividiríamos esta columna en 3 subcolumnas. Una nos indica los permisos del propietario (u) de ese archivo, la segunda los permisos del grupo (se refiere al grupo principal al que pertenece el propietario, se le asigna la letra g) y la última los permisos de el resto (o).



Si seguimos los permisos de la imagen, nos indica que el propietario tiene todos los permisos y tanto el grupo como el resto tienen de lectura(r) y de ejecución (x).

La asignación de permisos se puede realizar de dos formas. Asignación octal o simbólica.

* Asignación Octal: Para esta se asigna un valor a cada permiso, r = 4, w = 2 y x = 1. La asignación de permisos se hará sumando estos valores. EJ. Si quiero añadir los permisos lectura y escritura sumariamos 4 + 2 = 6.
  + Con el parámetro **-R** se asignan permisos a un directorio y a todo lo que este contenga.

Esto añadiría al propietario el permiso 6 = lectura y escritura. Al grupo 5 = lectura y ejecución. Al resto 1 = ejecución.

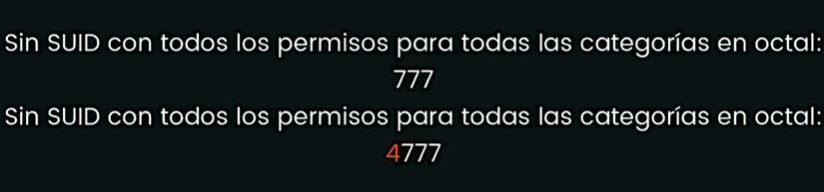
* Asignación simbólica: Usamos símbolos para los permisos **r** para lectura, **w** para escritura y **x** para ejecución. Y Símbolos para saber a quién vamos a poner el permiso. **U** para el propietario, **g** para el grupo y **o** para el resto. Por último, tenemos los operadores.
  + + 🡪 Agregamos permisos.
  + - 🡪 Quitamos p ermisos.
  + = 🡪 Dejamos los permisos tal cual se los indicamos.
  + a 🡪 de All, realiza el cambio a todas las categorías, propietario, grupo y otros. Ej. Para asignarles a todos lectura y escritura.

**PERMISOS SUID (Set User ID) Y SGID (Set Group ID):** Permisos especiales, adicionales a los que ya hemos visto. Permiten ejecutar ficheros con los permisos del propietario (SUID) o del grupo (SGID), es decir, es como si cualquier usuario pudiese simular tener los permisos del propietario o del grupo para acceder a un fichero. Esto implica grandes riesgos de seguridad y por ello es muy usado.

* **SUID**: Su uso principal sería el de evitar darle a un usuario el acceso completo a archivos importantes (como archivos del sistema protegidos). Se representa con el valor **s**. Y es un permiso que si se tiene figura sustituyendo el valor de ejecución (x) en el apartado del usuario propietario (u).



Su notación en Octal es el número 4 y no es incompatible con el valor de r que también es 4. Esto se debe a que van en posiciones distintas.



Para su asignación simbólica, sólo hay que añadir el símbolo **s** a los anteriores.

**Avanzado:** Se han encontrado vulnerabilidades en archivos con el privilegio SUID. Una forma de buscar archivos que tengan SUID es:

Con este comando buscaríamos en un directorio, archivos (la f es de files) que contengan el privilegio 4000 que es el SUID (-perm -4000).

* **SGID**: Es como el anterior, pero para grupos. Para directorios afecta de forma recursiva, es decir, todos los subdirectorios heredarían estos permisos.

Su representación simbólica también es **s**. Esto no es incompatible, puesto que en este caso se pone en la columna de grupo no de usuario.

En asignación octal tiene un valor de 2. El resto es igual que SUID.

Entonces si tenemos un valor de 4, antes de los permisos normales, es que tiene SUID, si tenemos un 2, es que tiene SGID y si tiene un 6 es que tiene ambos.

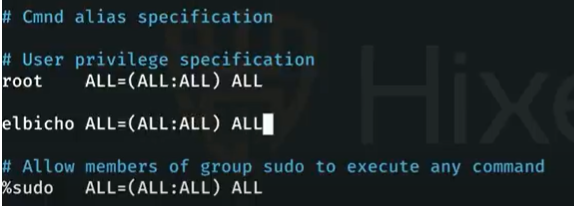
**Avanzado:** Como SUID también se pueden buscar archivos con SGID, sería lo mismo pero en este caso el número que usamos es 2000.

**PERMISOS SUDOERS:** En este interviene el comando sudo (el cual ya vimos anteriormente), el fichero sudoers (en el directorio /etc) que es el fichero que contiene las reglas para los usuarios sudoers, definiendo quien puede ejecutar que comandos y en qué condiciones. Es decir, el comando sudo se rige por las normas que hay en este fichero.

Este fichero es muy delicado y existen varias formas de editar este fichero, como con nano o VIM. Pero se recomienda hacerlo con **visudo** ya que cuando editamos con este comando lo que hace es bloquear el archivo para que no mas de una persona lo pueda editar a la vez, además proporciona verificación de la sintaxis básica evitando guardar cambios que podrían dejar el sistema inseguro o inactivo.

Indagaremos un poco en los campos que se pueden editar o agregar.

* + User: Define que usuario o grupo tiene el permiso sudo.
  + Host: Especifica las máquinas para las que esta regla es válida.
  + Alias: Definimos alias para usuarios, host y comandos.
  + Directivas:
    - **NOPASSWD:** Permite ejecutar comandos sin contraseña.
    - **PASSWD:** Requerirá contraseña para ejecutar comandos.
  + **All:** Representa a todos los campos, EJ. Si queremos otorgar a un usuario llamado elbicho permisos generales. Escribiríamos en el fichero:



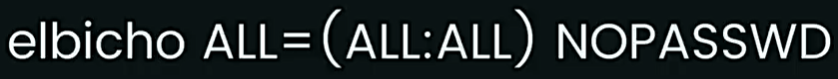


El primer all referente al Host, quiere decir que se aplica a todas las máquinas.

Los all entre paréntesis nos dicen que elbicho puede ejecutar los comandos como cualquier otro usuario (incluyendo el usuario root), o como cualquier grupo.

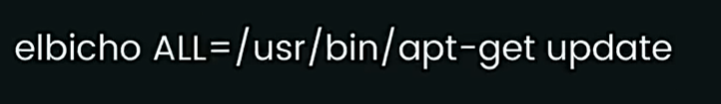
El último all especifica que comandos tiene derecho a ejecutar. En este caso todos los comandos.

Pongamos algunos ejemplos más:



En este caso el usuario el bicho puede ejecutar todos los comandos sin necesidad de contraseñas. Si antes era dios ahora es el dios de los dioses.

La regla No necesariamente tiene que tener todos los valores, los que quitemos no se aplican. EJ.



En este caso con el primer All nos indica que la regla es válida en todas las máquinas. Nos saltaríamos los valores del paréntesis y vamos directamente al campo del comando donde le decimos que sólo puede ejecutar el binario apt-get sólo con el comando update. Los valores que nos hemos saltado no se aplicarían.

**Grupo SUDO:** Los usuarios que pertenecen a este grupo tiene derechos elevados. Para añadir un usuario a este grupo se hace con el comando usermod, comando que explicamos anteriormente.

Donde a es añadir, G es Grupo (añadir grupo), sudo que es el grupo en cuestión y por último el usuario.

**PERMISOS STICKY BIT:** Sirve para controlar el acceso a ficheros en directorios compartidos. Esto es que cuando se aplica Sticky Bit en un directorio solamente el propietario de un fichero dentro del directorio puede manipular dicho fichero y se aplica incluso si otros usuarios tienen permisos de escritura dentro del directorio. Ejemplo:

Estoy con un usuario llamado Pepe, creo una carpeta llamada MisCosas con todos los permisos. Entro en el directorio y creo un archivo llamado lista.txt. que este archivo solo tiene permisos de lectura, pero no de escritura.

Ahora un usuario llamado Roberto puede entrar en mi directorio MisCosas y borrarme el archivo losta.txt puesto que aunque solo tenga permisos de lectura el archivo en sí, el directorio en el que se encuentra si que tiene habilitados todos los permisos.

Pues los permisos Sticky Bit sirven para que aunque la carpeta padre tenga los permisos el usuario Roberto no pueda borrar mi archivo lista.txt.

Su representación simbólica es una **t** que va en la columna de otros remplazando ejecución. y en octal se representa con el número **1** en la posición anterior a los permisos normales.

LSATTR: Sirve para ver permisos avanzados que tenga un archivo o directorio.

CHATTR: Sirve para asignar permisos avanzados.

Con +i ponemos el archivo como inmutable, ahora ni el usuario Root puede eliminarlo.

*Comandos para editar propietarios y grupos*

* **CHOWN:** Permite cambiar el propietario y el grupo de un fichero determinado.

Con esto asignaríamos un nuevo usuario y un nuevo grupo al archivo. Se puede solo hacer cambio de usuario. Para realizar solo un cambio de grupo usaremos el comando CHGRP.

* + –R 🡪 Realiza el cambio de forma recursiva.
* **CHGRP:** (Change Group) Nos sirve para cambiar únicamente el grupo.
  + –R 🡪 Realiza el cambio de forma recursiva.

*Capabilities*

Son propiedades que tu previamente puedes configurar en ciertos binarios para poder ejecutar determinadas tareas. En ciberseguridad nos pueden servir para escalar privilegios. Para ver los archivos que tienen capabilities utilizamos.

Para asignar una capabilitie utilizamos:

*Cron*

Cron es un administrador de tareas de Linux (se almacena por lo general en /etc/cron.d) que permite ejecutar comandos en un momento determinado, por ejemplo, cada minuto, día, semana o mes. Si queremos trabajar con cron, podemos hacerlo a través del comando **crontab**.

El formato de configuración de cron es el siguiente: **Minuto Hora Dia-del-Mes Mes Dia-de-la-Semana Comando-a-Ejecutar**

El intervalo de tiempo se especifica mediante 5 campos que representan, de izquierda a derecha:

* **Minutos**: de 0 a 59.
* **Horas**: de 0 a 23.
* **Día del mes**: de 1 a 31.
* **Mes**: de 1 a 12.
* **Día de la semana**: de 1 a 6 lunes a sábado (1=lunes, 2=martes, etc.) y 0 o 7 el domingo.

Si quisiéramos especificar todos los valores posibles de un parámetro (minutos, horas, etc.) podemos hacer uso del asterisco (**\***). Esto implica que, si en lugar de un número utilizamos un asterisco, el comando indicado se ejecutará cada minuto, hora, día de mes, mes o día de la semana, como en el siguiente ejemplo:

**\* \* \* \* \* /home/user/miscript.sh**

Enlace donde tenemos una calculadora cron.

<https://www.site24x7.com/es/tools/crontab/cron-generator.html>

*Conceptos de bash*

En Bash se pueden usar argumentos desde la línea de comandos, los cuales son enviados a los scripts como variables. Estos quedarían representados de la siguiente forma:

**[$0]**: Representa el nombre del script que se invocó desde la terminal.

**[$1]**: Es el primer argumento desde la línea de comandos.

**[$2]**: Es el segundo argumento desde la línea de comandos y así sucesivamente.

**[$#]**: Contiene el número de argumentos que son recibidos desde la línea de comandos.

**[$\*]**: Contiene todos los argumentos que son recibidos desde la línea de comandos, guardados todos en la misma variable.

Es decir, que si creamos un script donde pongamos por ejemplo echo $1, $1 lo tomará del primer argumento que le pasemos cuando ejecutemos el script.

El resultado será que imprimirá Hola por pantalla.

*Librerías útiles*

* Js-beautify -> Nos da formato a un archivo de JavaScript que venga en bruto sin formato alguno.