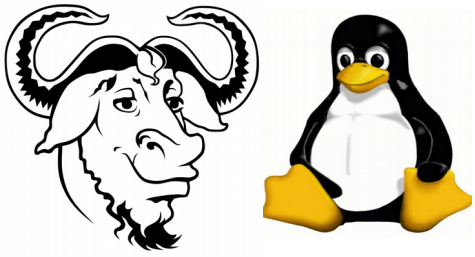
UD4 – ADMINISTRACIÓN DE GNU / LINUX



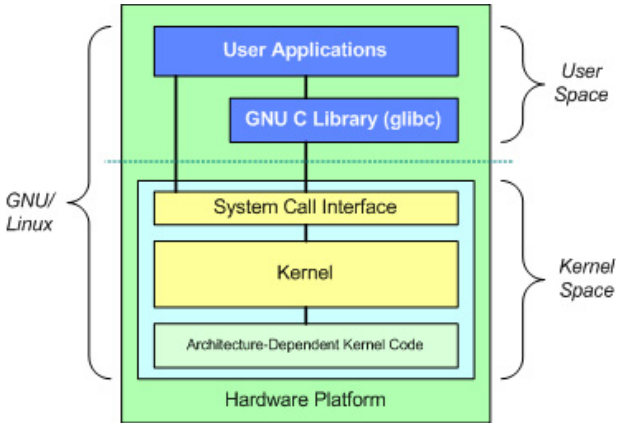
ÍNDICE

1. INTRODUCCIÓN

Los sistemas GNU/Linux cuentan con una amplia variedad de usuarios y de ámbitos de trabajo donde son utilizados. Su origen se remonta al mes de agosto de 1991, cuando un estudiante finlandés llamado **Linus Torvalds** anunció en una lista de news que había creado su propio núcleo de sistema operativo y lo ofrecía a la comunidad de desarrolladores para que lo probara y sugiriera mejoras para hacerlo más utilizable. Éste sería el origen del **núcleo** (o **kernel**) del sistema operativo que más tarde se llamaría **Linux.**

Por otra parte, la **FSF** (Free Software Foundation), mediante su **proyecto GNU**, producía software (desde 1984) que podía ser utilizado libremente. Debido a lo que **Richard Stallman** (miembro de la FSF) consideraba software libre, es decir, como aquél del que podíamos conseguir sus fuentes (código), estudiarlo, modificarlo y redistribuirlo sin que nos obliguen a pagar por ello. En este modelo, el negocio no está en la ocultación del código, sino en el software complementario añadido, en la adecuación del software a los clientes y en los servicios extras, como el mantenimiento y la formación de usuarios (el soporte que les demos), ya sea en forma de material, libros y manuales, o en cursos de formación.

La combinación (o suma) del **software GNU** y del **kernel Linux**, es el que nos ha traído a los actuales **sistemas GNU/Linux**. Actualmente, los movimientos Open Source, desde diferentes organizaciones (como FSF) y empresas como las que generan las diferentes distribuciones Linux (Red Hat, Mandriva, SuSe, Ubuntu...), pasando por grandes empresas como HP, IBM o Sun que proporcionan apoyo, han dado un empujón muy grande a los sistemas GNU/Linux hasta situarlos al nivel de poder competir, y superar, muchas de las soluciones propietarias cerradas existentes.



2. INSTALACIÓN

La instalación de GNU/Linux es muy similar a la de otros sistemas operativos, la mayoría de las distribuciones emplean **asistentes gráficos** para guiarnos durante el proceso de instalación. Muchas distribuciones también disponen de una opción comúnmente denominada **“Live CD”** que permiten probar el sistema arrancando desde un medio extraíble sin instalar absolutamente nada en el equipo, esta opción también resulta útil para restaurar sistemas y recuperar datos.

Cada distribución de GNU/Linux nos presenta diferente métodos para lograr su instalación, pero existe un número de pasos que son básicamente los mismos en todas:

- **Configuración Principal:** particiones y gestores de arranque.
- **Pre-configuración:** selección de paquetes o grupos de paquetes.
- **Instalación:** del kernel, utilidades básicas y paquetes seleccionados.
- **Post-configuración:** definición de la contraseña del root, crear cuentas y grupos de usuarios adicionales, configuración del sistema...

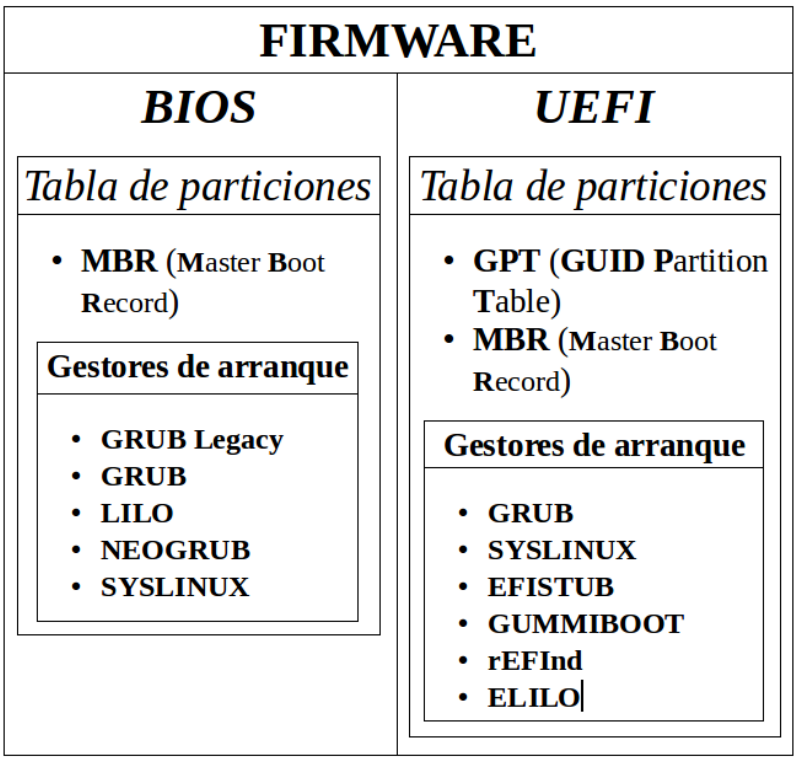
Existen más de 1600 distribuciones de GNU/Linux diferente. Los Kernel de todas las distribuciones de Linux comparten la misma línea evolutiva pero cada distribución tiene sus rutinas de instalación, gestor de paquetes, documentación específica, y una comunidad de soporte. Donde ellas difieren esencialmente es en su visión y características particulares para lograr tareas específicas. Algunas de las distribuciones principales son:

- **Debian:** Desarrollada por voluntarios y enfocada en la utilización de software verdaderamente OpenSource, fácil mantenimiento de los paquetes y actualización del sistema.
- **Ubuntu:** Versión depurada de Debian. El proyecto Ubuntu está patrocinado por Canonical Ltd, una compañía de holding fundada por el sudafricano Mark Shuttleworth.
- **Red Hat Enterprise:** Versión Red Hat Linux para empresas, su venta es manejada por una entidad comercial focalizada en soporte y compatibilidad.
- **CentOS:** Versión gratuita de Red Hat Enterprise.
- **Fedora:** Versión de Red Hat Linux no corporativa.
- **Slackware:** Una de las primeras distribuciones; conocida por el poco espacio que utiliza del disco duro.

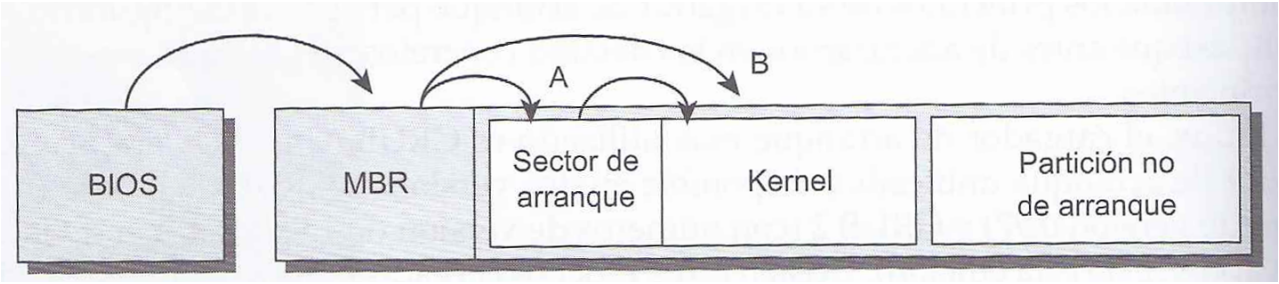
- **Gentoo**: Una distribución basada en código fuente, conocida por ser una distribución que es totalmente compilada paquete a paquete.
- **SuSE:** Distribución Alemana-Europea basada en paquetes RPM con orientación comercial y muy buen soporte, y versiones para una gran variedad de arquitecturas.
- **Turbolinux:** Distribución de gran acogida en Asia basada en RPM, acompañada de soluciones propietarias de high-end clustering (Nodos de Super Computadores) además de distribuir versiones de Servers y Estación de Trabajo (Workstation).
- **Linux Mint:** Linux Mint es una distribución del sistema operativo GNU/Linux, basado en la distribución Ubuntu, también hay una versión basada en Debian. Linux Mint viene con su propio juego de aplicaciones (Mint tools) con el objetivo de hacer más sencilla la experiencia del usuario.

Cual es la mejor distribución depende de las necesidades de cada usuario. Todas ellas tienen sus ventajas y desventajas particulares.

Durante el proceso de instalación, sea cual sea la distribución elegida, se configura el sistema para que se inicie GNU/Linux, si se tiene más de un sistema operativo instalado aparecerá un menú de elección. GNU/Linux instala código de arranque para que se inicie el gestor de arranque que carga el kernel GNU/Linux y se inicie el sistema operativo.



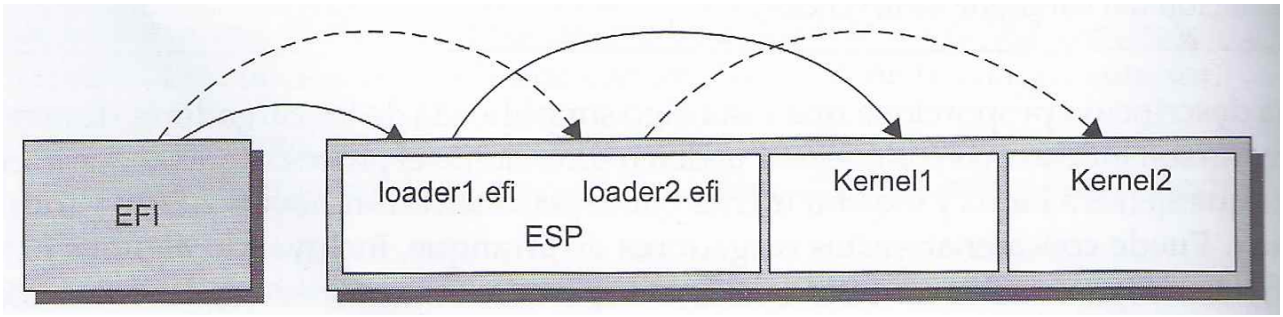
Si el disco utiliza **MBR**, el primer sector (512 Bytes) contiene la tabla de particiones y el código de arranque maestro (Master Boot Code), leído por la BIOS para continuar con el proceso de arranque. Este código de arranque busca en la tabla de particiones aquella partición que está marcada como activa, lee su sector de arranque y ejecuta el código de arranque de la partición para que continué con el inicio del sistema cargando el gestor de arranque.



Existen varios **gestores de arranque** que vienen con las distribuciones, uno de los más habituales es **GRUB** (Grand Unified Boot Loader), que se pueden utilizar para iniciar el sistema desde unidades externas o desde los discos duros. Su archivo de configuración principal es **/boot/grub/grub.cfg,** el cual **no es recomendable editar manualmente**, ya que este archivo es un archivo creado automáticamente por el sistema utilizando scripts que son los que se deben modificar para cambiar los ajustes de Grub2.

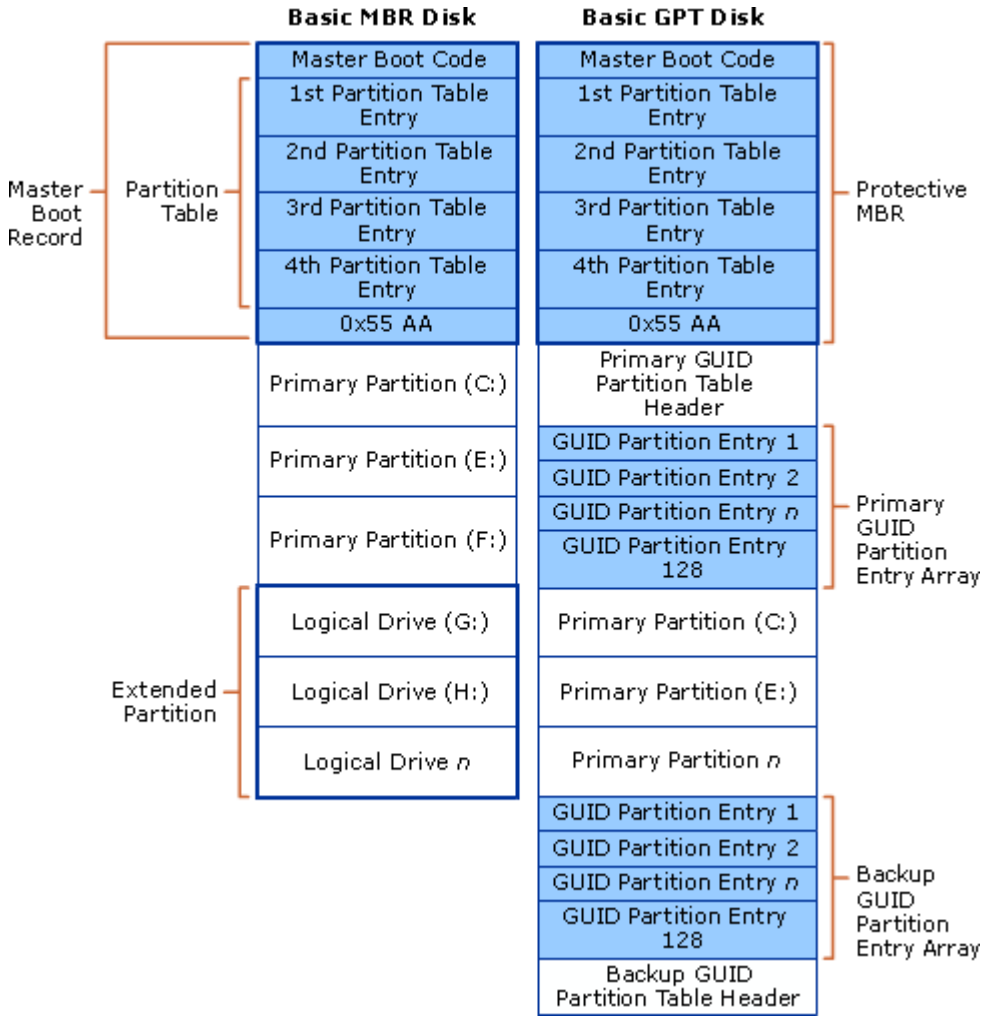
EFI permite el arranque tanto desde el MBR como desde GPT, el cual solventa las limitaciones técnicas de MBR. EFI posee su propio gestor de arranque minimalista que permite también la selección y carga directa del sistema operativo, eliminando la necesidad de recurrir a gestores de arranque externos.

El nuevo firmware EFI es mucho mas complejo que la antigua BIOS, en lugar de recurrir a código almacenado en sectores de arranque del disco duro, EFI recurre a **cargadores de arranque** almacenados como archivos en una partición del disco, denominada **ESP** (EFI System Partition) que usa el sistema de archivos FAT. En Linux el ESP se suele montar en **/boot/efi**. Los cargadores de arranque residen en archivos con la extensión .efi en subdirectorios que reciben el nombre del sistema operativo o del cargador de arranque (/boot/efi/EFI/ubuntu/grub.efi). Esta configuración le permite almacenar un cargador de arranque independiente para cada sistema operativo que se instale en el equipo. El firmware EFI incluye su propio programa, un administrador de arranque, para que pueda seleccionar el cargador de arranque secundario con el que iniciar un determinado OS.



Los equipos con MBR permiten un máximo de 4 particiones primarias (o 3 primarias y una extendida en la que crear más particiones lógicas), EFI nos permite hasta un máximo de 128 particiones (x86_64) todas primarias, pudiendo manejar particiones de más de 2TB. A parte de estas limitaciones hay que tener en cuenta las siguientes recomendaciones:

- Si se utiliza GRUB Legacy como gestor de arranque, es necesario utilizar MBR.
- Para un arranque dual con Windows que utiliza UEFI en lugar de BIOS, se debe usar GPT.
- Se recomienda siempre usar GPT con el arranque UEFI dado que algunos firmwares UEFI no permiten arrancar UEFI desde MBR.



Los **discos** son referidos por archivos en el directorio **/dev** con nombres muy estandarizados:

- **/dev/sda** Primer dispositivo SCSI o SATA
- /dev/sdb Segundo dispositivo SCSI o SATA
- /dev/sdc Tercer dispositivo SCSI o SATA
- etc..

Las **particiones** son nombradas utilizando la combinación de su dispositivo seguido por un número, en discos con MBR los números del 1 al 4 son reservados para las particiones primarias, aunque no se utilicen, o para una partición extendida, las particiones lógicas comienzan a enumerarse desde el 5:

- /dev/sda1 (Primera partición primaria)
- /dev/sda2 (Segunda partición primaria)
- /dev/sda3 (Tercera partición primaria)
- /dev/sda4 (Cuarta partición primaria)
- /dev/sda5 (Primera partición lógica)
- /dev/sda6 (Segunda partición lógica)
- etc..

El **espacio de particiones** más simple y básico para un sistema GNU/Linux requiere **una partición para el sistema de ficheros raíz /** nativa de Linux como **ext4** y una partición **swap** como **área de intercambio** de la memoria virtual, con el propósito de servir al sistema de un espacio de intercambio en almacenamiento secundario utilizado cuando la memoria RAM esta llena. Distribuciones más modernas también permiten utilizar un fichero dinámico en la raíz como área de intercambio, ya no siendo necesaria una partición independiente para la swap.

En el proceso de instalación nos podemos decidir también por alojar parte del sistema de archivos de GNU/Linux en **particiones independientes**. Algunas de las particiones que se pueden crear son las siguientes:

- **/usr** Contiene el software y comandos del sistema. Si se pone en una partición independiente debe ser lo suficientemente grande como para permitir que crezca si se instalan aplicaciones nuevas.
- **/boot** Contiene el kernel, los archivos necesarios para cargarlo y el gestor de arranque Grub.
- **/home** Directorio principal de los usuarios. Es recomendable crearlo en una partición independiente para evitar la perdida de datos.
- **/var** Contiene archivos de configuración y datos del sistema.
- **/tmp** Espacio de archivos temporales.

El archivo **/etc/fstab** es usado para definir cómo las particiones, o distintos dispositivos de bloques o sistemas de archivos remotos deben ser montados e integrados en el sistema durante el arranque. El archivo /etc/fstab contiene los siguientes campos:

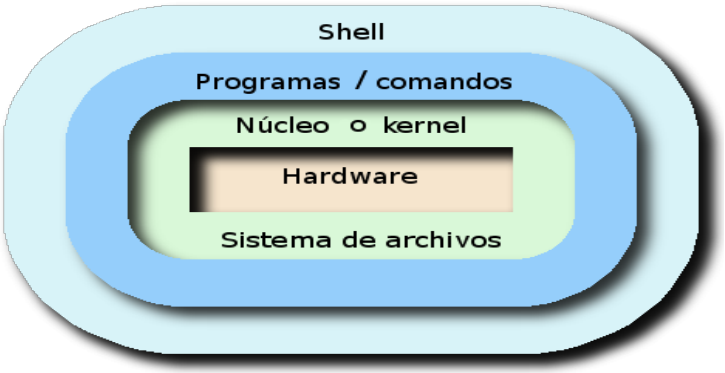
<file system> <dir> <type> <options> <dump> <pass>

- <file system> → Define la partición o dispositivo de almacenamiento para ser montado. Se puede usar el nombre de dispositivo o el UUID asociado que podemos consultar con *“sudo blkid”*.
- <dir> → Indica el punto de montaje (directorio) donde la partición será montada.
- <type> → Indica el tipo de sistema de archivos de la partición o dispositivo de almacenamiento para ser montado (ext3, ext4, nfs, vfat, ntfs,..)

- <options> → Indica las opciones de montaje que la orden mount utilizará para montar el sistema de archivos. La opción “defaults” para ext4 son: rw (lectura y escritura), suid (permite las operaciones de setuid y setgid), dev (dispositivo de bloque), exec (permite la ejecución de binarios), auto (montado automático), nouser (solo el usuario root puede montar y desmontar el sistema de archivos) y async (todo el I/O se debe hacer de forma asíncrona).
- <dump> → Utilizado por el programa dump (volcado) para decidir cuándo hacer una copia de seguridad. Dump comprueba la entrada en el archivo fstab y el número de la misma le indica si un sistema de archivos debe ser respaldado o no. La entradas posibles son 0 y 1. Si es 0, dump ignorará el sistema de archivos, mientras que si el valor es 1, dump hará una copia de seguridad. La mayoría de los usuarios no tendrán dump instalado, por lo que deben poner el valor 0 para la entrada <dump>.
- <pass> → Utilizado por fsck para decidir el orden en el que los sistemas de archivos serán comprobados. Las entradas posibles son 0, 1 y 2. El sistema de archivos raíz (/) debe tener la más alta prioridad: 1, todos los demás sistemas de archivos que desea comprobar deben tener un 2. La utilidad fsck no comprobará los sistemas de archivos que vengan ajustados con un valor 0 en <pass>.

3. ESTRUCTURA DE GNU / LINUX

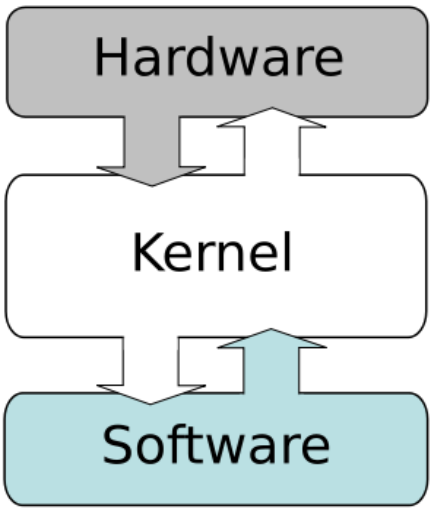
Para comprender el funcionamiento del sistema GNU / Linux es necesario comprender su estructura. Este sistema operativo está formado por varios componentes principales. Entre ellos, el **Núcleo** (o Kernel), el **Shell**, el **sistema de archivos**, los **programas** y los **comandos**.



3.1. El Kernel

El **Kernel** es la parte del sistema operativo que sirve para **interactuar con el hardware**. Proporciona una serie de servicios que pueden ser utilizados por los programas, sin que éstos tengan que preocuparse de cómo se gestiona el hardware.

En general, el **núcleo** es el encargado de gestionar la memoria , mantener el sistema de archivos, del manejo de interrupciones, manejo de errores, realización de los servicios de entrada/salida, asignación de los recursos de la CPU, comunicación entre procesos, etc..



El kernel se carga en memoria cuando el sistema se inicia y **permanece en memoria** hasta que el sistema se descarga por completo. Se diseña para ser lo más pequeño que sea posible, permitiendo así que la memoria restante sea compartida entre todos los programas que se ejecutan en el sistema. Los programas se relacionan con el ordenador a través del núcleo mediante las **llamadas al sistema.**

El Kernel suele estar escrito en **C**, con algo de lenguaje **ensamblador** que le facilita trabajar directamente con el hardware. Una de las ventajas de los sistemas Linux es que, al estar el código fuente del kernel disponible, es más sencillo desarrollar nuestros propios controladores y módulos del kernel.

El código ejecutable del núcleo de Linux (el que se carga en memoria al arrancar el ordenador) reside en un fichero denominado **vmlinux** (o en su homólogo comprimido, **vmlinuz**). Normalmente, este fichero se encuentra en el directorio **/boot** y suele tener como sufijo la versión del núcleo que generó el ejecutable. El **número de versión** asociado al kernel tiene un sentido muy particular ya que está ligado a su desarrollo.

La versión del núcleo Linux actualmente consta de cuatro números **A.B.C[.D]:**

- El número **A** denota la versión del núcleo. Es el que cambia con menor frecuencia y solo lo hace cuando se produce un gran cambio en el código o en el concepto del núcleo. Históricamente sólo ha sido modificado cuatro veces: en 1994 (versión 1.0), en 1996 (versión 2.0), en 2011 (versión 3.0) y en 2015 (versión 4.0).
- El número **B** denota la subversión del núcleo. Antes de la serie de Linux 2.6.x, los números pares indicaban la versión estable y los números impares versiones de desarrollo. Comenzando con la serie Linux 2.6.x, no hay diferencia entre los números pares o impares.
- El número **C** indica una revisión del núcleo. Con la nueva política, solo es cambiado cuando se introducen nuevos drivers o características; cambios menores se reflejan en el número **D.**
- El número **D** se utiliza cuando un error grave requiere de un arreglo inmediato y no hay otros cambios como para lanzar una nueva revisión. Bug-fixes y parches de seguridad son actualmente manejados por el cuarto número dejando los cambios mayores para el número **C**.

Otro fichero muy relacionado con el kernel es **initrd** o el disco RAM inicial, que es un sistema de archivos temporal usado por el núcleo Linux durante el inicio del sistema. Es usado típicamente para hacer los arreglos necesarios antes de que el sistema de archivos raíz / pueda ser montado.

El kernel original, sin las modificaciones de las distribuciones, incluyendo su código fuente, se puede conseguir en *http://www.kernel.org* (The Linux Kernel Archives).

3.2. El Shell

El **Shell** es la parte que permite al usuario comunicarse con el sistema. Puede estudiarse el shell desde dos puntos de vista: como **intérprete de comandos** y como **lenguaje de programación**, que combina mediante estructuras de control grupos de comandos almacenados en archivos llamados **Shell Scripts** o procedimientos Shell.

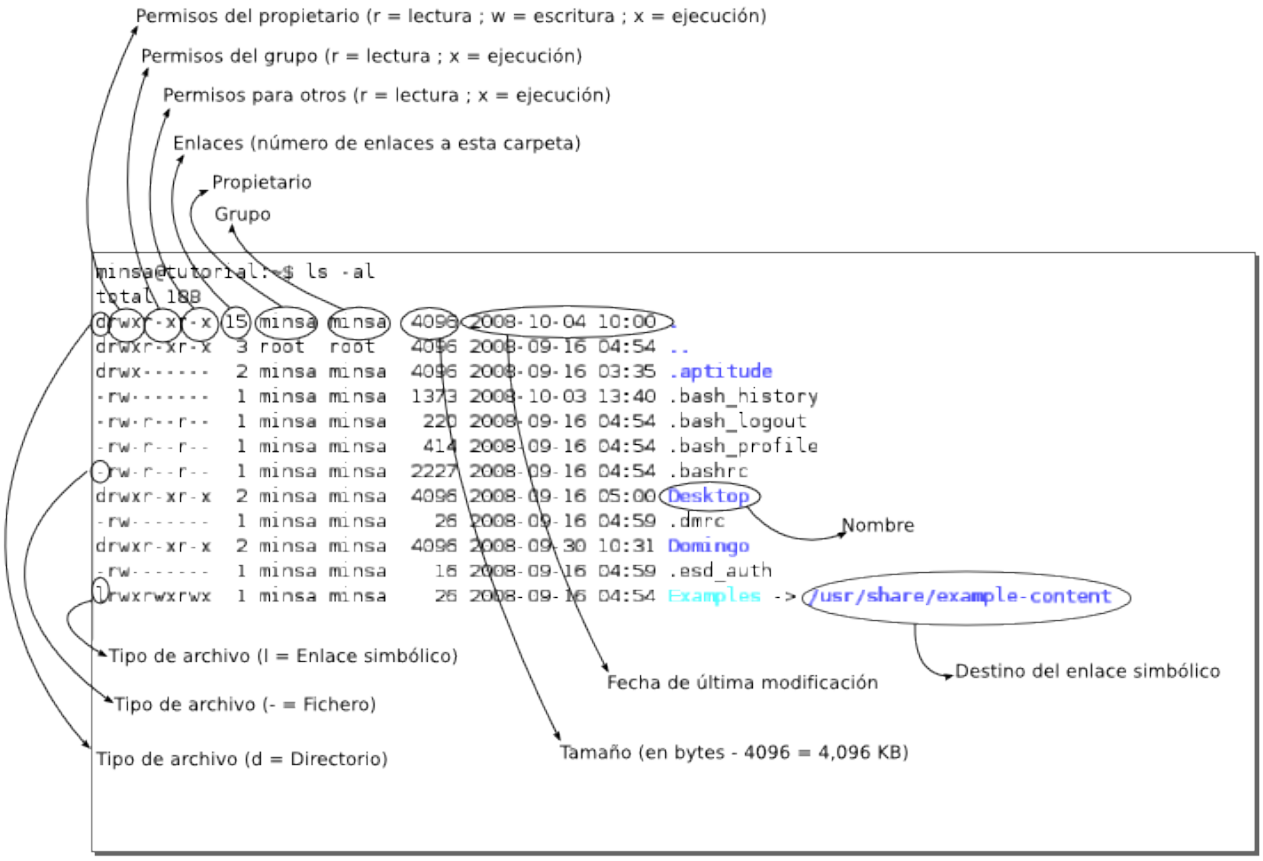
Cuando el usuario introduce un comando, el Shell, que es un programa en continua ejecución, analiza la línea y llama al programa o programas que realiza la función solicitada por el comando. Existen diferentes interpretes de comandos:

- **SH**: Shell Bourne. Su indicativo por defecto es el símbolo dólar "$". Este Shell está capacitado para redireccionar la salida y entrada estándar, interpretar los metacarácteres, manejar variables y usar tuberías y filtros.
- **CSH**: C-Shell. Tiene todas las características del SH, pero añade algunas específicas para los programadores de C. Su prompt de sistema queda representado por el símbolo porcentaje “%”.
- **BASH**: Bourne Again Shell. Este shell es una ampliación del SH, puesto que, entre otras características, permite edición de comandos ejecutados, tamaño ilimitado del histórico de comandos, control de trabajos y procesos, funciones, alias, cálculos aritméticos, etc.. Este shell es el más utilizado en Linux.
- **Otros**: KSH: Korn shell. TCSH: Enhanced C Shell. ZSH: Z Shell. JSH: Shell Job..

El administrador del sistema debe adjudicar uno de estos shells a cada usuario. Éste, a su vez, puede ejecutar cualquier shell siempre y cuando tenga autorización para ello. Cuando un usuario se conecta al sistema se inicia automáticamente un programa de shell denominado **shell de presentación.** Este shell se carga de forma automática cuando se accede al fichero **/etc/passwd,** que contiene información de los usuarios incluido el shell estándar que usará cada uno.

3.3. El sistema de archivos

Linux define una interfaz abstracta al nivel del kernel que incorpora varios sistemas de archivos diferentes. Linux define siete tipos de archivos, para determinar cuál es el tipo de un archivo se utiliza el comando ***ls –l.*** El primer carácter de la salida que genera ***ls*** determina el tipo de archivo.



Tipo de archivo	Símbolo
Archivos comunes	-
Directorios	dd
Archivos de los dispositivos de caracteres	cc
Archivos de los dispositivos de bloque	bb
Conexiones del dominio local	ss
Cauces designados	pp
Enlace simbólico	l

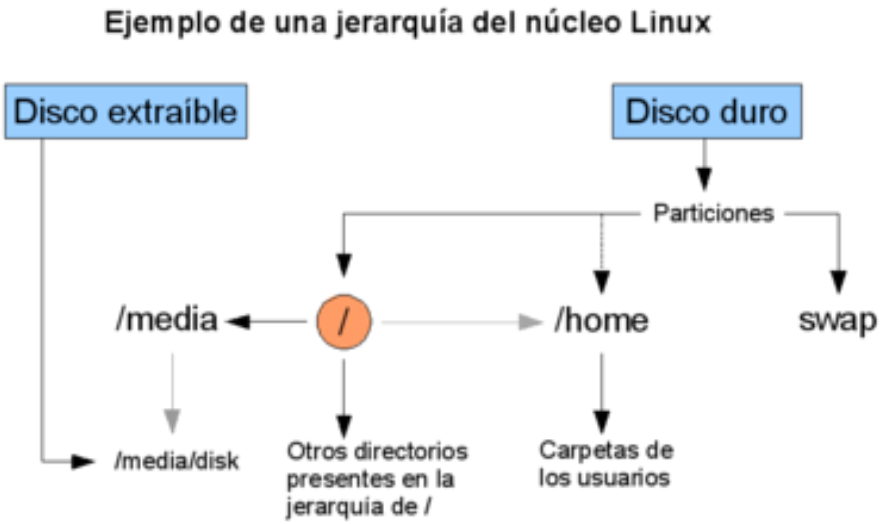
Un **archivo común** es un conjunto de bytes y Linux no impone una estructura determinada a su contenido. En Linux los **archivos ocultos** comienzan con un punto y para listarlos se debe incluir el parámetro **–a** en la orden ls, así el comando más utilizado para listar los ficheros es ***ls -la.***

Los **archivos de dispositivo** permiten que el hardware del sistema se comunique con los periféricos. Los archivos de los dispositivos están caracterizados por dos números, el mayor le dice al kernel a qué controlador hace referencia el dispositivo y el menor le indica al controlador la unidad física a la que se ha de dirigir. Los **dispositivos orientados a bloques** transmiten datos en bloques utilizando el buffer de datos del sistema operativo y los **dispositivos orientados a caracteres** transmiten solo un bit o un solo Byte a la vez sin hacer uso del buffer.

Las **conexiones del dominio local** son conexiones que se establecen entre procesos en red y que permiten intercambiar información sin interferencias. Los **cauces designados** permiten que se establezca una comunicación entre dos procesos que se están ejecutando en el mismo ordenador.

La diferencia entre los **enlaces físicos** y los **enlaces simbólicos** es que los enlaces físicos son archivos comunes que referencia directamente al contenido del archivo enlazado y los enlaces simbólicos son una referencia al nombre del archivo enlazado.

Un **directorio** es un fichero que contiene referencias a otros archivos, es interesante conocer la estructura principal de directorios de los sistemas GNU/Linux, ya que aunque pueden haber diferencias según la distribución utilizada, la mayoría siguen el estándar de jerarquía de sistemas de archivos o **FHS** (Filesystem Hierarchy Standard), que define los directorios principales y sus contenidos.



El contenido de algunos de los directorios más importantes se resume en la siguiente tabla:

Ruta	Contenido
/	Directorio raíz del sistema
/bin	Comandos y programas esenciales
/boot	Kernel y archivos necesarios para cargarlo (grub2)
/dev	Todos los dispositivos físicos del sistema (discos, impresoras..)
/etc	Archivos de configuración e inicio
/home	Directorio principal de los usuarios
/lib	Librerías y partes del compilador de C
/media	Unidades físicas montadas (discos duros, DVDs, pen drives..)
/mnt	Sistema de archivos montados temporalmente
/opt	Paquetes de software para aplicaciones opcionales
/proc	Información sobre los procesos cargados
/root	Directorio principal del superusuario
/sbin	Comandos para iniciar, reparar y recuperar el sistema
/srv	Datos servidos por el sistema
/sys	Área de trabajo del kernel y archivos de configuración
/tmp	Archivos temporales
/usr	Datos de usuario, contiene la mayoría de las utilidades y aplicaciones
/var	Archivos variables, como logs y temporales

3.4. Programas y comandos

No es objetivo de estos apuntes explicar todos los programas y todos los comandos con sus parámetros posibles para los sistemas GNU/Linux. El uso de la **ayuda del sistema** debería bastar para este fin. Una de las principales habilidades que debe desarrollar un Administrador de Sistemas, consiste en usar correctamente la ayuda. Cualquier sistema que usemos contará con al menos un nivel de ayuda, que debemos saber buscar e interpretar.

La documentación de Linux se encuentra dispersa en una gran variedad de fuentes, las más importantes son:

- Las páginas de ayuda del comando **man** (se pueden instalar las páginas de ayuda en castellano con "*sudo apt install manpages-es manpages-es-extra"*)
- Documentos Texinfo accesibles con el comando **info**
- Ayuda breve de un comando añadiendo el parámetro **-h**, **--help** ó **-?**
- Ayuda para los comando integrados en el shell con **help**
- Descripción breve de un comando con **whatis**
- Localizar los fuentes, binarios y páginas del manual para los programas que se especifiquen con **whereis**
- Obtener la ruta de un ejecutable con **which**
- Buscar todo lo relacionado con una palabra con **apropos**
- Documentación propia de cada distribución (ej. *https://help.ubuntu.com*)
- En Internet de forma general existen multitud sitios sobre GNU/Linux

Para interpretar correctamente la ayuda debemos conocer su sintaxis, normalmente se siguen los siguientes convenios:

- **texto en negrita** : se debe teclear exactamente igual.
- *texto en cursiva :* hay que reemplazar esto por el argumento apropiado.
- [-abc] : uno o todos los argumentos entre corchetes son opcionales.
- -a | -b : las opciones separadas por | no pueden usarse conjuntamente.
- argumento … : el argumento es repetible.
- [expresión] … : la expresión entre corchetes completa es repetible.

```
man  [-c|-w|-tZT  dispositivo] [-adhu7V] [-m sistema[,...]] [-L locale]
[-p cadena] [-M  ruta]  [-P  paginador]  [-r  prompt]  [-S  lista]  [-e
extension] [[sección] pagina ...] ...
```

4. RESUMEN DE COMANDOS BÁSICOS

4.1. Manipulación de ficheros y directorios

ls: permite listar el contenido de un directorio.

Sintaxis: ls [opciones] [directorio | fichero]

Algunas opciones:

- -l : muestra la salida en formato largo.
- -R: lista recursivamente un directorio.
- -a : lista además los ficheros ocultos (comienzan con punto).
- -h : muestra el tamaño de los ficheros en forma más legible.
- -i : muestra el identificador del i-nodo asociado a cada elemento.

cd: se utiliza para cambiar el directorio actual.

Sintaxis: cd [directorio]

Ejecutando **cd** a solas cambia al directorio home del usuario, con **cd ..** retrocedemos al directorio anterior, con **cd /** nos situamos en la raiz del sistema de ficheros.

pwd: indica el camino absoluto del directorio actual.

mkdir: se utiliza para crear directorios.

Sintaxis: mkdir [opciones] [directorio]

La opción -p permite crear directorios intermedios si es necesario.

mv: mueve un fichero hacia otro, o varios ficheros hacia un directorio. Este permite a su vez renombrar ficheros o directorios.

Sintaxis: mv [opciones] <origen> <destino>

Algunas opciones:

- -i : ejecuta el comando de forma interactiva, es decir, pregunta antes de sobrescribir el destino si existiera.
- -u : actualiza (upgrade) el destino con el fuente solo si este es más reciente.

cp: permite copiar un fichero en otro, o varios ficheros en un directorio.

Sintaxis: cp [opciones] <origen> <destino>

Algunas opciones:

- -R : copia recursivamente un directorio.
- -i : pregunta antes de sobrescribir el destino.
- -l : hace enlaces fuertes a los ficheros fuentes en lugar de copiarlos.

ln: permite crear vínculos con otros ficheros.

Sintaxis: ln [-s] <fichero apuntado> <nombre del enlace>

El vínculo creado por defecto se denomina **enlace duro** (hard link) porque directamente crea el enlace al inodo del fichero enlazado. La opción –s crea un **enlace simbólico**, es decir, un enlace al nombre del fichero enlazado.

rm: se utiliza para borrar (desenlazar) ficheros

Sintaxis: rm [opciones] <ficheros | directorios>

Algunas opciones:

- -r : borra recursivamente un directorio.
- -f :fuerza el borrado sin detenerse ante errores.
- -i : ejecuta el comando de forma interactiva.

stat: muestra las características de un fichero.

Sintaxis: stat [opciones] <fichero>

diff: permite comparar ficheros de texto línea a línea, y nos indica en caso de que no sean iguales en que líneas cambian. Dado que en Linux se realizan muchas configuraciones del sistema modificando ficheros de texto, es una orden interesante para saber si algo se ha modificado.

Sintaxis: diff fichero1 fichero2

du: permite conocer la longitud de una jerarquía de ficheros a partir de un directorio.

Sintaxis: du [opciones] [ficheros | directorios]

Algunas opciones:

- -h: (human readable view) formato más legible.
- -s: suma el tamaño de cada fichero/directorio sin profundizar recursivamente en estos últimos.
- -c: produce un total cuando se utilizan varios argumentos.

touch: permite crear ficheros vacíos.

Sintaxis: touch [opciones] [nombre_fichero]

4.2. Visualización y edición de ficheros

cat: concatena ficheros y los imprime en la salida estándar. Si no se le pasa ningún argumento lee de la entrada estándar.

vi: editor estándar de Unix y está orientado a comandos. Existe una versión conocida como **vim** (Vi IMproved) muy poderosa que permite la edición de múltiples ficheros, edición automática para varios lenguajes de programación, ayuda en línea, selección visual, varios niveles de deshacer, etc. Para algunos usuarios, vi resulta incómodo pues para utilizar toda su potencia es necesario conocer muchas combinaciones de teclas, pero si se llega a dominar resulta muy funcional. Su principal virtud es que encontraremos vi en prácticamente cualquier versión de Unix que usemos, cosa que no se puede decir de otros editores (joe, pico, edit, gedit, nano, emacs, etc.).

Básicamente vi posee dos modos de interacción: el de inserción (edición) y el de comandos. Para pasar al **modo comando** se pulsa **Esc** y para pasar al **modo inserción** se pulsa **i**.

Algunos comandos útiles en vi (pulsando ESC para pasar al modo de comandos, que veremos en la última línea de la pantalla).

- dd - borra la línea actual.
- D - borra desde la posición actual hasta el final de la línea.
- dG - borra hasta el final del fichero.
- u - deshace el último comando.
- :q - sale del editor (si se han hecho modificaciones y no se ha guardado se genera un error).
- :q! - sale sin guardar.
- :w - guarda.
- :wq - guarda y sale.
- :x - guarda y sale.
- <n><comando> - ejecuta el comando n veces.
- / texto – busca el texto en el fichero. Poniendo luego solo una / indica buscar siguiente.

more y **less:** Los comandos more y less paginan (dividen en páginas) uno o varios ficheros y los muestran en pantalla. Se diferencian en las facilidades que brindan. Por ejemplo more es más restrictivo en cuanto al movimiento dentro del texto, mientras que less no limita este aspecto pues acepta el empleo de todas las teclas de movimiento tradicionales. Cuando se alcanza el final del último fichero a paginar, more termina automáticamente, no así less. También more muestra sucesivamente el

porcentaje del fichero visto hasta el momento. Tanto less como more proveen una serie de comandos para moverse con facilidad dentro del texto paginado.

Algunas teclas que podemos usar mientras usamos estos programas son:

- q: permite interrumpir el proceso y salir.
- /p: realiza búsquedas del patrón p dentro del texto. Para repetir la búsqueda del mismo patrón sólo es necesario escribir /.
- [n]b: en more permite regresar n páginas (por defecto n es 1).
- [n]f: en more se adelantan n páginas y en less, n líneas.

file: determina con cierto grado de precisión el tipo de un fichero que se le pasa como argumento.

Sintaxis: file [opciones] [ficheros | directorios]

4.3. Búsquedas de ficheros y patrones

Caracteres comodines: permiten al usuario nombrar los archivos de forma abreviada y acceder a muchos en un solo comando.

- **?**: Cuando este carácter es utilizado como parte del nombre de un archivo o directorio, la interrogación sustituye a un solo carácter.
- *****: Cuando es utilizado como parte del nombre de un archivo o directorio, el asterisco sustituye a cualquier carácter o grupo de caracteres.
- **[ ]**: Cuando son utilizados como parte de nombres de archivos o directorios, representan un solo carácter de los incluidos en la posición donde estén, es decir, uno solo de los caracteres puestos entre corchetes. Se pueden poner rangos separados por un guión.
- **!**: Operador negación. Permite la exclusión de caracteres. No es un metacarácter.

grep: busca patrones en ficheros. Por defecto devuelve todas las líneas que contienen un patrón (cadena de texto) determinado en uno o varios ficheros. Utilizando las opciones se puede variar mucho este comportamiento. Si no se le pasa ningún fichero como argumento hace la búsqueda en su entrada estándar.

Sintaxis: grep [opciones] <patrón> [ficheros]

Algunas opciones:

- -c: devuelve sólo la cantidad de líneas que contienen al patrón.
- -i: ignora las diferencias entre mayúsculas y minúsculas.

- -H: imprime además de las líneas, el nombre del fichero donde se encontró el patrón. Es así por defecto cuando se hace la búsqueda en más de un fichero.
- -l: cuando son múltiples ficheros sólo muestra los nombres de aquellos donde se encontró al patrón y no las líneas correspondientes.
- -v: devuelve las líneas que no contienen el patrón.
- -r: busca en un directorio de forma recursiva.
- -n: imprime el número de cada línea que contiene al patrón.

find: permite buscar de forma recursiva en un directorio todos los ficheros que cumplan ciertas condiciones. Las condiciones pueden estar relacionadas con el nombre de los ficheros, el tamaño, los permisos, el tipo, las fechas de acceso y modificación, etc.

Sintaxis: find [camino] [opciones]

Algunas opciones:

- -name <expresión> permite especificar patrones para los nombres de los ficheros a buscar.
- -iname <expresión> permite especificar patrones para los nombres de los ficheros a buscar sin tener en cuenta mayúsculas y minúsculas.
- -type <tipo> permite indicar el tipo de fichero a buscar. Este puede ser d para directorios, f para ficheros regulares, l para enlaces simbólicos, b para dispositivos de bloque, c para dispositivos de carácter, p para tuberías y s para sockets.
- -size +/-<n> permite indicar el tamaño máximo y/o mínimo de los ficheros a buscar. Por defecto el tamaño se expresa en bloques de 512 bytes, pero si se precede este por un carácter c se referirá a bytes, k a kilobytes, w a palabras de dos bytes y b a bloques.
- -perm [+|-]<modo> permite referirse a aquellos ficheros cuyos permisos sean exactamente modo, incluya todos los de modo (signo -) o incluya alguno de los de <modo> (signo +). El valor de <modo> se expresa en forma numérica.
- -exec <comando> ; permite definir un comando a ejecutarse para cada resultado de la búsqueda. La cadena { } se sustituye por el nombre de los ficheros encontrados. El caracter ; permite indicar la finalización del comando. (Tanto { } como ; tienen que ir entre comillas o entre contrabarras para evitar que sea sustituido por el shell).

sed: permite borrar líneas, registros o sustituir cadenas de caracteres dentro de las líneas. Uno de los usos más interesantes de sed es sustituir cadenas. Podemos sustituir una cadena por otra de la siguiente manera:

sed -i 's/cadena1/cadena2/' fichero

Al ejecutar el comando anterior, se sustituye la primera cadena que encuentra por la segunda. Pero, si lo que queremos es sustituir todas las cadenas que encuentre, en cada una de las líneas, añadimos el parámetro g:

sed -i 's/cadena1/cadena2/g' fichero

locate: busca en una base de datos, actualizada periódicamente, todos los paths en la jerarquía de ficheros que contengan una cadena determinada. Para crear esta base de datos o actualizarla se debe invocar por root el comando updatedb.

4.4. Redireccionamiento, tuberías y filtros

Redireccionamiento de entrada (comando < fichero): este símbolo (<) significa que tome la entrada para un comando desde el siguiente archivo, en lugar de tomar la entrada estándar (teclado).

Redireccionamiento de entrada (comando << texto): este símbolo (<<) significa que tome la entrada para un comando desde la entrada estándar hasta que se escriba una palabra igual a la especificada en “texto”.

Redireccionamiento de salida (comando > fichero): este símbolo (>) significa que redireccione al archivo siguiente la salida estándar de un comando.

Redireccionamiento de adición (comando >> fichero): el símbolo >> opera de la misma forma que el símbolo >, con la única diferencia que el resultado del comando es "añadido" a continuación del texto que tuviera el archivo especificado al lado derecho del símbolo. Si no existe el archivo de salida lo crea.

Redireccionamiento de errores (2> , 2>>): el símbolo 2> significa que cualquier mensaje de error producido se almacene en el archivo especificado a continuación del símbolo, en lugar de usar la pantalla. En Unix existe un dispositivo ficticio /dev/null

(dispositivo nulo) que se puede utilizar cuando una salida o un error no queremos que aparezca en ningún dispositivo o fichero.

Redireccionamiento de salida (comando &> fichero): este símbolo (&>) significa que redireccione al archivo siguiente la salida estándar y la salida de error de un comando.

Tuberías: (|): Para enlazar los comandos entre sí, se utiliza el carácter (|). La salida proporcionada por un comando es tomada por el siguiente comando como entrada. La cantidad de comandos en una tubería o pipeline es, en principio, ilimitada. Si no es suficiente una sola línea de comandos puede ponerse al final el carácter "\" y continuar en la línea siguiente.

sort: ordena las líneas de un fichero mostrándolas por la salida estándar. De no especificarse un fichero toma la entrada estándar.

Sintaxis: sort [opciones] [fichero]

Algunas opciones:

- -r : ordena al revés.
- -f : trata las mayúsculas y minúsculas por igual.
- -g : ordena de forma numérica, de modo que no es necesario que los números se rellenen con ceros por la izquierda.

uniq: elimina las líneas repetidas de un fichero ordenado, imprimiéndolo por la salida estándar o en otro fichero. De no especificarse un fichero toma la entrada estándar.

Sintaxis: uniq [opciones] [fichero] [salida]

Algunas opciones:

- -c : utiliza como prefijo en cada línea su número de ocurrencias.
- -d : solo imprime las líneas duplicadas.

tail y **head:** muestran respectivamente el final y el comienzo (10 líneas por defecto) de uno o varios ficheros. De no especificarse al menos un fichero toman la entrada estándar.

Sintaxis: tail/head [opciones] [ficheros]

Algunas opciones:

- -f: para el caso de tail se ejecuta de forma sostenida o sea se continúa visualizando el final del fichero hasta que se interrumpa el proceso (Ctrl-c).
- -q: no coloca los encabezamiento con el nombre de los ficheros cuando se indican varios (quiet).

- -<n>: imprime las n últimas (primeras) líneas en lugar de las diez establecidas por defecto.

wc: imprime el número de líneas, palabras y bytes de uno o varios ficheros. Si son varios ficheros hace también un resumen de los totales. De no especificarse un fichero toma la entrada estándar.

Sintaxis: wc [opciones] [ficheros]

Algunas opciones:

- -l sólo cuenta líneas.
- -c sólo cuenta bytes.
- -w sólo cuenta palabras.

cut: permite cortar una línea de texto, para obtener un subconjunto en lugar de la línea completa. Podemos cortar por número de caracteres, por campos, etc.

Sintaxis: cut [opciones] [ficheros]

Algunas opciones:

- -c N-M: corta desde el carácter número N hasta el carácter número M.
- -c N-: corta desde el carácter número N hasta el final
- -c –N: corta desde el principio hasta el carácter número N
- -c N,M: corta el carácter número N y el carácter número M
- -d":" –f1: separa la línea en campos divididos por el carácter : y nos muestra sólo el primer campo
- -d"-" –f3: separa la linea en campos divididos por el carácter – y nos muestra sólo el 3 campo.

4.5. Compactar y agrupar ficheros

gzip y **gunzip:** permiten compactar y descompactar (comprimir y descomprimir) respectivamente uno o varios ficheros.

Sintaxis: gzip [opciones] <ficheros/directorio>

Algunas opciones:

- -r: dado un directorio comprime todos los ficheros presentes en él recursivamente.
- -1 a -9 : especifica el grado de la compresión (-1 menor y más rápida -9 mayor y más lenta).
- -S <sufijo> : permite especificar un sufijo o extensión para el fichero resultado (por defecto es gz).

tar: es una herramienta para agrupar varios ficheros aislados o el contenido de un directorio en otro fichero o dispositivo especial. El comando tar no comprime o compacta absolutamente nada, se limita a agrupar varios ficheros en uno solo, sin comprimirlos. Existe una opción (-z) que automáticamente ejecuta un gzip o un gunzip sobre el fichero agrupado.

Sintaxis: tar [opciones] <fuentes>

Algunas opciones:

- -c: permite crear (tarear), es decir, agrupar ficheros en uno solo.
- -x: permite extraer (destarear), es decir, desagrupar ficheros.
- -v: activa el modo debug, donde se ven todos los mensajes.
- -f <fichero>: agrupa o desagrupa en o hacia un fichero y no utilizando la salida o entrada estándar como es por defecto. (Ojo, esta opción la usaremos siempre).
- -z: compacta o descompacta el fichero resultante una vez agrupado o desagrupado con gzip y gunzip respectivamente.
- -t: lista el contenido de un fichero resultado de un agrupamiento.
- -M: agrupa en volúmenes.

El comando tar conserva la estructura jerárquica original de lo agrupado excluyendo el carácter / que representa a la raíz. Algunas opciones se pueden emplear sin el carácter -, siempre y cuando no haya ambigüedades entre ellas o con los argumentos.

4.6. Comandos de dispositivos y del sistema

uname: muestra información del sistema.

fsck: chequea y repara un sistema de ficheros.

fdisk: permite crear y modificar las particiones, con la opción –l muestra todas las particiones del sistema. También existen aplicaciones gráficas como **gparted**.

Sintaxis: fdisk dispositivo

cfdisk: versión avanzada de fdisk, permite crear particiones, eliminarlas y redimensionarlas.

mkfs: crea un sistema de ficheros en un dispositivo (formatea).

mkisofs: permite crear imágenes iso de dispositivos ópticos.

mount: monta un dispositivo establecido en /etc/fstab.

Sintaxis: mount [opciones] dispositivo punto_de_montaje

umount: desmonta un dispositivo.

df: informa del uso de disco en los distintos volúmenes montados.

Sintaxis: df <opciones>

El parámetro –h muestra la información de forma más legible.

chroot: establece el directorio raíz y permite ejecutar un proceso bajo un directorio raíz simulado, de manera que el proceso no puede acceder a archivos fuera de ese directorio.

Sintaxis: chroot <directorio> <comando>

lshw y **dmidecode:** lista el hardware del equipo.

free: informa del consumo de memoria del sistema, tanto en memoria real como en memoria virtual (swap).

alias: permite asignarle otros nombres a los comandos del sistema.

Sintaxis: alias [nombre[=valor]...]

Sin argumentos muestra todos los alias definidos por el usuario actual. Para deshabilitar un alias se emplea el comando **unalias**. Para que un alias esté siempre disponible lo podemos incluir en el fichero **~/.bashrc**.

history: muestra los últimos comandos ejecutados en el sistema con -c borra el historial.

lsmod: muestra el estado de los módulos del kernel cargados actualmente.

modprobe: permite administrar los módulos del kernel.

lspci: lista los dispositivos PCI del sistema.

lsusb: lista los dispositivos usb del sistema.

blkid: muestra los atributos de las particiones del sistema.

uptime: muestra el tiempo que lleva encendido el sistema y otros datos.

dmesg: lista el buffer de mensajes del núcleo.

vmstat: muestra información del sistema como memoria, swap, cpu, etc.

4.7. Comandos de redes

ping: permite enviar paquetes ICMP (Internet Control Message Protocol) del tipo ECHO_REQUEST a otro dispositivo, con el objetivo de saber si es alcanzable a través de la red. Además muestra un resumen estadístico acerca del tanto por ciento de paquetes perdidos y las velocidades de transmisión. Este comando se ejecuta por defecto ininterrumpidamente por lo que para pararlo se debe hacer Ctrl-c.

Sintaxis: ping [opciones] <máquina>

Algunas opciones:

- -c <n>: envía n paquetes exactamente.
- -i <n>: espera n segundos entre los envíos.
- -s <n>: envía paquetes de n bytes. Se le suman los 8 bytes del header del paquete ICMP.
- -q: sólo despliega el sumario final.

tracepath: permite trazar una ruta entre nuestro host y el host que le indiquemos, mostrándonos todos los pasos intermedios que va recorriendo.

mtr: combinación avanzada de ping y tracepath.

ssh: permite conectarnos a un ordenador remoto de forma segura desde el shell.

Sintaxis: ssh [opciones] usuario_remoto@host_remoto

scp: permite copiar ficheros de o desde máquinas remotas.

scp usuario@host:directorio/ArchivoOrigen ArchivoDestino

scp ArchivoOrigen usuario@host:directorio/ArchivoDestino

write: se utiliza para enviar un mensaje a un usuario conectado al sistema. Por defecto el mensaje se envía a la última terminal donde se haya conectado el usuario. Los usuarios pueden deshabilitar la posibilidad de recibir mensajes utilizando el comando **mesg**.

Sintaxis: write <usuario> [terminal]

wall: se emplea para enviar un mensaje a todos los usuarios conectados en el sistema que tengan habilitada la posibilidad de recibirlos (mesg y).

netstat: muestra información sobre las conexiones de red, las tablas de enrutamiento, las estadísticas de los interfaces de red, etc. Con este comando podemos conocer perfectamente que esta ocurriendo en nuestra red y como se comporta nuestro equipo en la misma.

route: gestiona la tabla de enrutamiento del sistema.

wget: permite descargar ficheros de la red (incluido Internet) de forma no interactiva. wget permite descargar mediante los protocolos http, ftp, ssh, etc. Algunas opciones interesantes son:

- --tries=numero: establece el número de reintentos para descargar algo
- --limit-rate=numk: establece una velocidad máxima de descargas en KB
- -c: continua una descarga que se interrumpió anteriormente
- --user=usuario: indica un nombre de usuario para entrar en el sitio de descargas
- --password=contra: indica una contraseña para entrar en el sitio de descargas
- --no-directories: no replica la estructura de directorios
- -r –recursive: descarga recursivamente, buscando en los subdirectorios del sitio
- -l numero: indica en cuantos niveles de recursividad entrará la descarga
- -A: una lista de archivos a descargar, separados por coma. Se puede usar el nombre del fichero completo, una extensión, una parte del nombre, etc.

nslookup: permite interrogar a un servidor DNS para encontrar información sobre un dispositivo de la red.

dig: obtiene información sobre consultas y servidores DNS.

who: muestra los usuarios conectados al sistema ya sea local o remotamente.

Sintaxis: who [opciones] [fichero] [am i]

Sin argumentos who muestra los logins de los usuarios conectados, por que terminal lo han hecho y en que fecha y hora.

Algunas opciones:

- -H: imprime un encabezamiento para las columnas.
- -w: indica si está activada o no la posibilidad de recibir mensajes por parte de cada usuario conectado (+ indica que si, - que no y ?, desconocido).
- -i: imprime además para cada usuario conectado que tiempo lleva sin interactuar con el sistema (idle time). Si lleva menos de un minuto pone un punto y si es más de 24 horas la cadena ``old''.
- -q: sólo muestra los logins de los usuarios conectados y la cantidad total de ellos.

w: muestra también los usuarios conectados al sistema además de lo que están haciendo (proceso que ejecutan en ese momento) y otras informaciones.

Sintaxis: w [opciones] [usuario]

Sin argumentos este comando muestra una primera línea con: la hora en que arrancó el sistema, cuanto tiempo lleva funcionando, cuantos usuarios hay conectados (sin incluir las sesiones gráficas) y carga de la CPU: durante el último, los 5 y los 15 minutos anteriores. A continuación se muestra una tabla cuyas columnas representan: el login de cada usuario conectado, por que terminal lo ha hecho, desde que host, a que hora, el idle time exacto, la cantidad de segundos de CPU que han empleado todos los procesos que ha ejecutado ese usuario (JCPU) y el tiempo (PCPU) y nombre del comando que ejecuta actualmente.

finger: permite buscar y mostrar información asociada a los usuarios del sistema de acuerdo a sus nombres, apellidos o login. La información que muestra finger para cada usuario es:

- El login.
- El nombre y los apellidos.
- El directorio base.
- El shell.
- La oficina y el teléfono.
- El teléfono de la casa.
- La lista de terminales a través de las que está conectado con la fecha, tiempo sin interactuar (iddle time) y si está deshabilitada la posibilidad de recibir mensajes.
- La fecha y hora del último nuevo mensaje electrónico recibido y desde cuando no accede al buzón.
- El contenido del fichero .plan en el directorio base.

ifconfig: permite configurar por parte de root las interfaces de red. Los usuarios distintos de root lo pueden invocar también con fines informativos. Sin argumento ifconfig despliega información acerca de la configuración y funcionamiento actuales de las interfaces de red activas.

ifup y **ifdown:** habilita y deshabilita respectivamente los interfaces de red indicados.

iwconfig: similar a ifconfig pero para dispositivos wireless.

ip: comando que reemplaza a ifconfig y iwconfig, por ejemplo para mostrar la configuración de red se puede utilizar “ip a” o “ip addr”.

hostname: devuelve el nombre de equipo (host) y nos permite cambiarlo.

dhclient: solicita una nueva configuración de red al servidor dhcp.

arp: resuelve equivalencias entre direcciones IP y direcciones MAC.

nbtscan: muestra estadísticas de NetBIOS y conexiones de NetBIOS sobre TCP/IP.

nmap: es un escáner de puertos de red. Su principal función es comprobar una serie de dispositivos para ver qué puertos TCP y UDP están abiertos.

Sintaxis: nmap [opciones] host

Algunas opciones:

- -sT: para intentar conectar con todos los puertos TCP del ordenador que se está analizando.
- -O: intenta averiguar que sistema operativo está ejecutando el host remoto.
- -sV: intenta averiguar que aplicación está utilizando el puerto abierto.

tc: permite gestionar el control de tráfico, limitando las capacidades de una interfaz o generando prioridades de tráfico (QoS).

iptables: permite indicarle las reglas de filtrado que deseemos al firewall Netfilter de Linux.

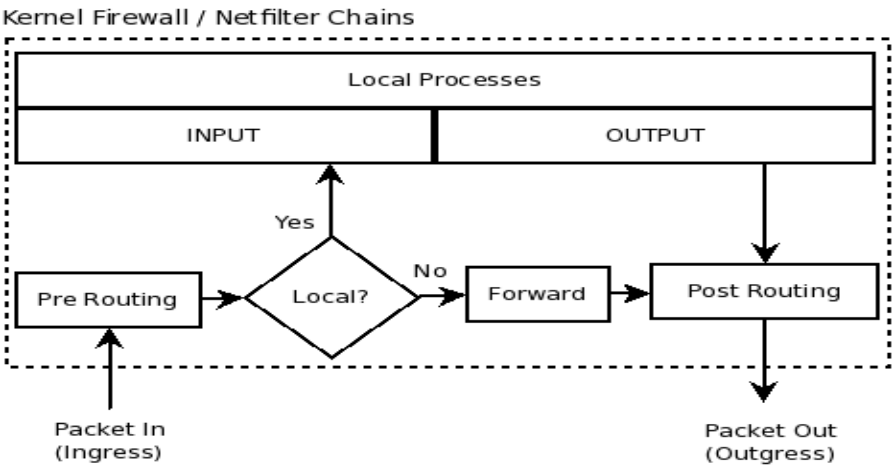
Sintaxis: iptables -t [tabla] -[opciones] [regla] [criterio] -j [acción]

Algunas opciones:

- -t [tabla] Especifica cuál es la tabla en la que aplicamos la regla. Existen 3 tipos básicos de tablas: FILTER, NAT y MANGLE. Filter, opción por defecto, es la tabla donde se añaden las reglas relacionadas con el filtrado de paquetes. Nat se refiere a las conexiones que serán modificadas por el firewall. Mangle es parecido a Nat, pero tiene la posibilidad de modificar más valores del paquete.
- -[opciones] Las opciones básicas del comando son las siguientes:
  - A para añadir (Append) una regla.
  - L es para listar (List) las reglas.
  - F es para borrar (Flush) todas las reglas.
  - P establece la política (Policy) por defecto del firewall. Por defecto es aceptar todas las conexiones.
- [regla] Reglas válidas son:
  - INPUT, para paquetes destinados al propio equipo.

- FORWARD, para paquetes generados en el equipo.
- OUTPUT, para paquetes que se deben encaminar.

- [criterio] Es donde se especificarán las características del paquete. Algunos ejemplos son:
  - -s: dirección de origen (source). Puede ser una dirección IP o una red.
  - -d: dirección de destino.
  - -p: tipo de protocolo (TCP,UDP,ICMP).
    - --sport: puerto de origen.
    - --dport: puerto de destino
  - -i interface: el interfaz por el que se entra.
  - -o interface: el interfaz por el que se sale.
- -j [acción] Aquí establecemos qué es lo que hay que hacer con el paquete. Las posibles acciones son:
  - ACCEPT: aceptar el paquete.
  - REJECT: desechar el paquete emitiendo un paquete ICMP Port Unreachable, indicando que está cerrado el puerto
  - DROP: descartar el paquete silenciosamente.
  - REDIRECT: redirigir el paquete a donde se indique en el criterio del comando.
  - LOG: archivar el paquete para su posterior análisis.

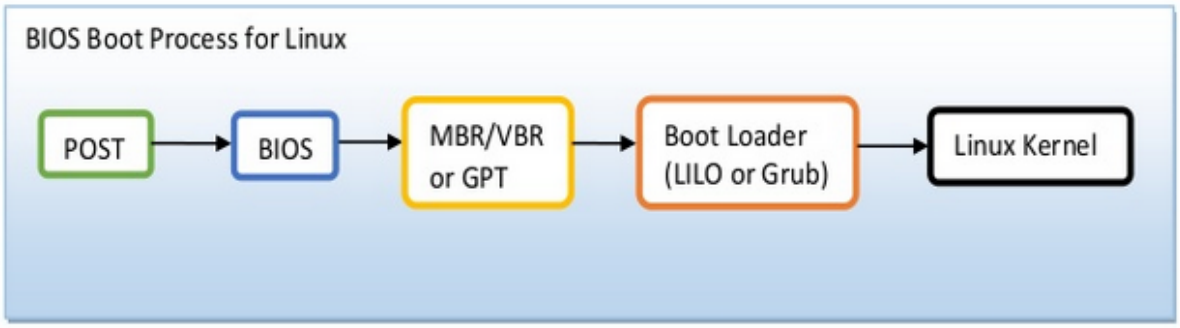


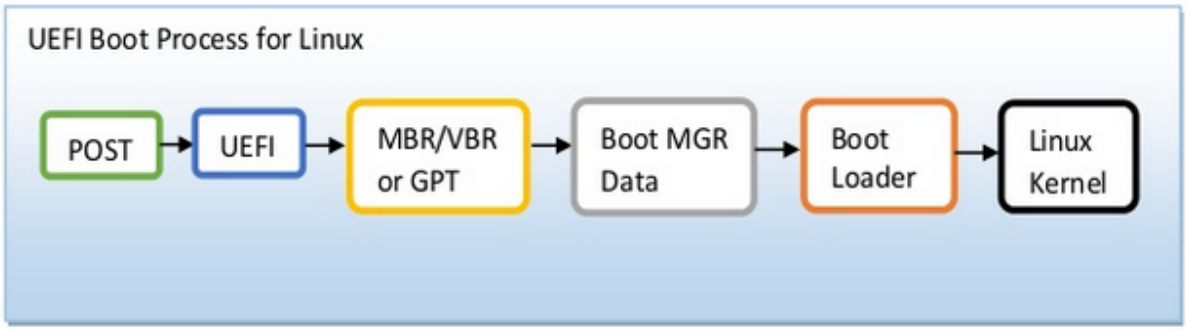
5. INICIAR Y CERRAR EL SISTEMA

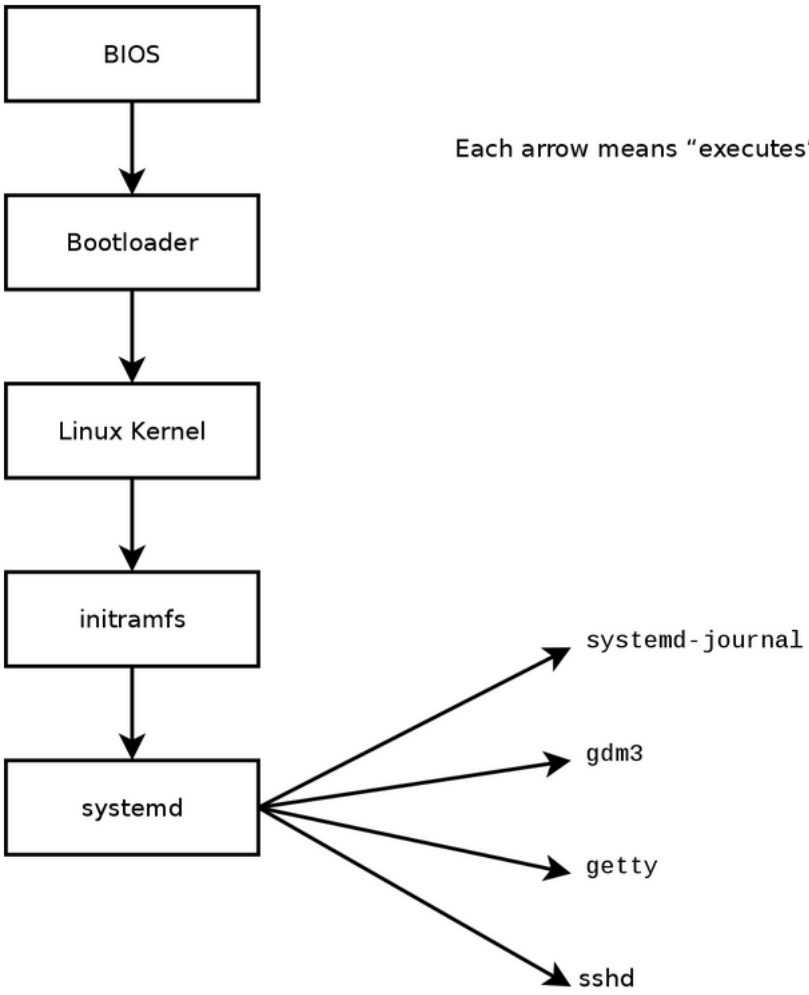
5.1. Proceso de arranque

En GNU/Linux, un proceso de arranque suele constar de los siguientes pasos:

1. Ejecución de las tareas de comprobación del hardware, **POST** (Power On Selft Test) por parte del **firmware** BIOS o UEFI.
2. La BIOS o UEFI carga el **MBR** (Master Boot Record) o el **GPT** (GUID Partition Table), leyendo los primeros sectores del disco duro, o el **VBR** (Volume Boot Record) leyendo el primer sector de una partición.
3. Si se usa BIOS, el MBR, VBR o GPT carga el **gestor de arranque** (GRUB2).
4. Si se usa UEFI se accede a la partición EFI y se carga el **cargador de arranque** por defecto (Boot MGR Data) desde \EFI\BOOT\boot{arch}.efi o bien el cargador de arranque o shell que tenga configurado el firmware. Este cargador de arranque lanza el **gestor de arranque** del sistema operativo (GRUB2).
5. Cuando el control pasa del firmware al gestor de arranque (GRUB2), se carga un **kernel.**
6. Cuando el kernel de Linux toma el control, este descomprime la imagen **initramfs** (sistema de archivos RAM inicial), que se convierte en el sistema de ficheros root inicial, y comienza la carga de dispositivos, monta la partición raíz y por último carga y ejecuta el programa inicial del sistema (por defecto /etc/init).
7. El **programa inicial** recibe el ID de proceso 1 (PID1) que es el primer programa que se ejecuta en el sistema de arranque tradicional, /sbin/init lee el archivo /etc/inittab para determinar que otros programas ejecutar. En sistemas que utilicen Upstart o systemd, /sbin/init lee otros archivos de configuración e incluso su nombre se ve reemplazado como es el caso para systemd donde init pasa a ser systemd
8. init lanza las **getty**, una para cada terminal virtual, las cuales inicializan cada tty pidiendo el nombre de usuario y contraseña (se puede cambiar de tty presionando ctrl+alt+[F1-F7]). Una vez que se proporcionan el nombre de usuario y la contraseña, getty los compara con las que se encuentran en /etc/passwd, llamando a continuación al programa login, para que, una vez iniciada la sesión de usuario, lance el shell de usuario de acuerdo con lo definido en /etc/passwd.


BIOS Boot Process for Linux
POST
BIOS
MBR/VBR or GPT
Boot Loader (LILO or Grub)
Linux Kernel


UEFI Boot Process for Linux
POST
UEFI
MBR/VBR or GPT
Boot MGR Data
Boot Loader
Linux Kernel


BIOS
Bootloader
Linux Kernel
initramfs
systemd
Each arrow means “executes”.
systemd-journal
gdm3
getty
sshd

5.2. Gestor de arranque. GRUB2

El kernel de Linux es un programa y lo primero que se hace al iniciar el sistema es cargarlo en memoria para que se pueda ejecutar. La ruta del kernel suele ser **/boot/vmlinuz.** El encargado de cargar el kernel en memoria es el **gestor de arranque,** existen varios gestores de arranque, siendo el más habitual en los sistemas GNU/Linux **GRUB** (GRand Unifier Bootloader).

GRUB2 (GRand Unifier Bootloader version 2) es el gestor de arranque predeterminado de algunas de las últimas versiones de Linux. Para modificar el tiempo de espera, sistema operativo por defecto, el nombre de los sistemas operativos y toda la información del arranque de cada uno de ellos (igual que se hacía antes en /boot/grub/menu.lst) se puede hacer mediante el archivo **/boot/grub/grub.cfg,** aunque no es recomendable hacerlo de este modo, ya que este archivo es un archivo creado automáticamente por el sistema utilizando otros archivos que son los que se deben modificar para cambiar los ajustes de Grub2, estos ficheros son:

- Los contenidos en el directorio /etc/grub.d/
  - /etc/grub.d/00_header: genera la cabecera del fichero grub.cfg indicando por ejemplo la imagen de fondo del grub, si se usa autenticación en el grub, etc..
  - /etc/grub.d/10_linux :contiene comandos y scripts que se encargan del kernel de linux en la partición principal.
  - /etc/grub.d/30_os-prober: contiene comandos y scripts que se encargan de buscar y añadir las entradas en el grub de otros sistemas operativos.
  - /etc/grub.d/40_custom: contiene entradas personalizadas.
- El archivo **/etc/default/grub** que contiene la configuración genérica del grub, como por ejemplo las siguientes:
  - GRUB_DEFAULT=0 indica la entrada del grub que se seleccionará por defecto que puede ser un número que expresa el orden de aparición en el menú o se puede indicar el nombre completo de una entrada.
  - GRUB_TIMEOUT=10 indica el tiempo de espera para que el usuario seleccione una entrada, en este caso 10 segundos.
  - GRUB_DISABLE_OS_PROBER="true" indica que no se ejecute el script /etc/grub.d/30_os-prober que añade automáticamente los sistemas operativos no Linux que se encuentren en el disco.

Una vez modificado cualquier fichero de configuración del grub, para que los cambios tengan efecto, es necesario indicarle al sistema que regenere el fichero **/boot/grub/grub.cfg** ejecutando el comando "**sudo update-grub2**"**.**

Para **recuperar el gestor de arranque** de Linux se puede utilizar el comando **grub-install,** en caso de que no sea posible iniciar el sistema debido a que el grub está dañado se pueden seguir los siguientes pasos:

1. Iniciar el sistema con un LiveCD de Linux

2. Crear un directorio vacío, por ejemplo:
   mkdir /media/disco1

3. Montar la partición donde está instalado realmente Linux en el directorio creado anteriormente:
   mount /dev/sda1 /media/disco1

4. Conectar el directorio /dev del LiveCD por el de nuestro sistema:
   mount --bind /dev /media/disco1/dev

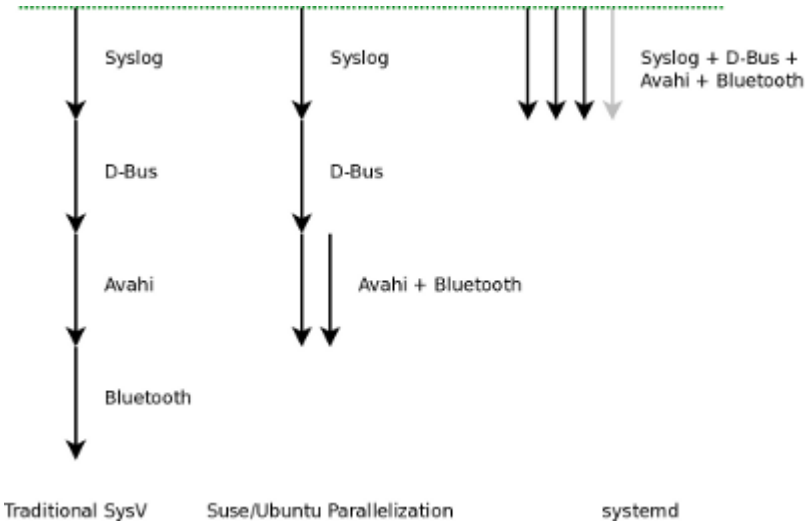
5. Indicar al sistema que el directorio montado es el directorio raíz:
   chroot /media/disco1

6. Realizar los comandos que necesitemos, en este caso para reinstalar el grub:
   grub-install /dev/sda

5.3. Proceso init. Systemd

En los sistemas UNIX el primer proceso que se crea y se encarga de inicializar el resto de procesos se le conoce como **init**. Desde 1980 esta tarea era llevada a cabo por **System V init** (sysvinit). Sin embargo, dado que en sysvinit los servicios se lanzan en un orden determinado y los dispositivos actuales son muy diversos y dinámicos está dejando de ser adecuado para estos tiempos. En el 2006 Canonical desarrollo **Upstart** con el objetivo de crear un reemplazo para sysvinit. Upstart es un sistema dirigido por eventos de tal forma que cuando ocurre cierta circunstancia se hace alguna acción. **Systemd** fue lanzado en 2010 por Lennart Poettering, también como un reemplazo de sysvinit. Systemd está comenzando a reemplazar a sysvinit como el proceso encargado de arrancar el sistema en varias distribuciones.

Hasta ahora el PID 1 era para el programa init, cosa que ha cambiado en systemd a favor de /usr/lib/systemd/systemd y además systemd al igual que Upstart deja de utilizar el archivo /etc/inittab. **Systemd** ha sido creado para ofrecer un inicio mas rápido y flexible que SysV, permitiendo el arranque paralelo de servicios.



Algunas de las mejoras que ofrece systemd son:

- Se ha mejorado sustancialmente la velocidad de inicialización del sistema.
- Asume que cualquier dispositivo puede ser conectado o desconectado en cualquier momento (hotplug).
- Utiliza la activación de daemons por medio de sockets, aportando capacidades de paralelización.

- Una de sus características es el registro (journal) mediante cgroups (grupos de control) de todos los servicios y procesos iniciados, permitiendo controlar la asignación de recursos a estos servicios y procesos durante el arranque.
- Es modular, esto quiere decir que se han elaborado una serie de “paquetes” en los que varios servicios son administrados de forma conjunta

Systemd inicia y supervisa todo el sistema y se basa en la noción de **unidades de servicios** (units), compuestas de un nombre (el nombre del demonio) y una extensión. Será la extensión la que indique de que tipo de unidad se trata. Además cada unidad tiene su correspondiente archivo de configuración cuyo nombre es idéntico. Un ejemplo sería el servicio httpd.service cuyo archivo de configuración sería httpd.service. Los archivos de unidades disponibles en nuestro sistema podemos encontrarlos en /lib/systemd/system/ y /etc/systemd/system/. Los archivos bajo el directorio /etc/systemd/system/ prevalecerán en caso de duplicados.

Existen siete tipos diferentes de unidades (units):

- service: Demonios que pueden ser iniciados, detenidos, reiniciados o recargados.
- socket: Esta unidad encapsula un socket en el sistema de archivos o en Internet. Cada unidad socket tiene una unidad de servicio correspondiente.
- device: Esta unidad encapsula un dispositivo en el árbol de dispositivos de Linux.
- mount: Esta unidad encapsula un punto de montaje en la jerarquía del sistema de archivos.
- automount: Encapsula un punto de montaje automático. Cada unidad automount tiene una unidad mount correspondiente, que se inicia al acceder al directorio de automontaje.
- target: Utilizada para la agrupación lógica de unidades. Referencia a otras unidades, que pueden ser controladas conjuntamente, un ejemplo sería multi-user.target, que básicamente desempeña el papel de nivel de ejecución 3 en el sistema clásico SysV.
- snapshot: Similar a las unidades target.

Entonces los archivos de configuración tendrán los nombres: programa.service, socket.socket, dispositivo.device, puntodemontaje.mount, etc…

Systemd al igual que Upstart ofrece compatibilidad con SysV (comando service y chkconfig) para aquellos servicios que aún soportan o funcionan únicamente con scripts de inicio SysV. Upstart pese a mantener compatibilidad con los comandos service y chkconfig de SysV implementó su propia utilidad para la administración de servicios initctl, de igual modo que systemd lo hace con su herramienta systemctl

systemctl stop nombreservicio.service

En SysV habilitábamos servicios con chkconfig (o reproducíamos listas de estos para ver cual de ellos se ejecutaba al arranque), algo que ahora bajo systemd podemos hacer con los siguientes comandos:

Habilitar el servicio httpd al arranque del sistema:

systemctl enable httpd.service

Listar todas las unidades de servicios instaladas:

systemctl list-unit-files

Podemos apreciar que a la hora de pasar el nombre del servicio lo hacemos con el nombre completo, es decir incluyendo su sufijo, pero existen una serie de atajos:

- Si no se especifica el sufijo, systemctl asumirá que es .service.
- Los puntos de montaje se traducirán automáticamente en la correspondiente unidad .mount. Por ejemplo, si especifica /home será equivalente a home.mount.
- Similar a los puntos de montaje, los dispositivos se traducen automáticamente en la correspondiente unidad .device, por lo tanto, la especificación /dev/sda2 es equivalente a dev-sda2.device.

Acción	systemd	SysV
Arrancar un servicio	systemctl start foo	service foo start
Detener un servicio	systemctl stop foo	service foo stop
Reiniciar un servicio	systemctl restart foo	service foo restart
Recargar el archivo de configuración de un servicio (en systemd no todos los servicios lo soportan)	systemctl reload foo	service foo reload
Rearrancar un servicio que ya se encuentra en ejecución	systemctl condrestart foo	service foo condrestart
Mostrar el **estado** del servicio	systemctl status foo	service foo status
Activar un servicio para que sea ejecutado durante el arranque	systemctl enable foo	chkconfig foo on
Desactivar un servicio para que no sea ejecutado durante el arranque	systemctl disable foo	chkconfig foo off
Muestra el **estado** de un servicio durante el **arranque**	systemctl is-enable foo (1º)	chkconfig --list foo
Crear o **modificar** el **archivo** de **configuración** de un servicio	systemctl daemon-reload	chkconfig --add foo
Listar los **modos de ejecución** para los que un servicio está activado o desactivado	ls /etc/systemd/system/*.wants/foo.service	chkconfig

Systemd utiliza **target** en vez de runlevels (0 1 2 3 4 5 6) que reciben un nombre (en vez de un número) para identificar un propósito específico, con la posibilidad de realizar más de una acción al mismo tiempo. Algunos targets pueden heredar todos los servicios de otro target e implementarse con servicios adicionales. La siguiente tabla muestra la similitud entre algunos de los target con los runlevels de SysV:

Nivel de ejecución sysvinit	Target systemd	Notas
0	runlevel0.target, poweroff.target	Detener el sistema.
1, s, single	runlevel1.target, rescue.target	Modo monousuario.
3	runlevel3.target, multi-user.target	Multiusuario, no gráfico. Los usuarios pueden usualmente hacer login por medio de múltiples consolas o desde la red.
5	runlevel5.target, graphical.target	Multiusuario, gráfico. Usualmente todos los servicios del nivel de ejecución 3 sumando el login gráfico.
6	runlevel6.target, reboot.target	Reiniciar el sistema.
emergency	emergency.target	Shell de emergencia.

Existen otros target que podremos ver con el comando:

systemctl list-units --type=target

Podemos cambiar de target (o modo de ejecución) actual con el comando:

systemctl isolate graphical.target

Systemd también nos permite cambiar el target predeterminado e incluso añadir nuevos servicios a otros target, pero antes de esto, es importante dejar claro algunos de los directorios de los que hace uso systemd:

- Los archivos unit (archivos de configuración de service, mount, device…) se encuentran en: /lib/systemd/system/ o /etc/systemd/system/.
- Los target (runlevels) igualmente pueden situarse en ambos directorios.
- Los directorios *.wants (ubicados igualmente en ambos directorios) contienen los enlaces simbólicos que apuntan a determinados servicios, serán estos servicios los que se ejecuten con el target al que corresponde dicho directorio wants, recordar que si un target precisa de otro, también serán cargados los servicios de este otro target.

El target /lib/systemd/system/default.target es el target predeterminado de arranque, es un enlace simbólico que por defecto apunta a /lib/systemd/system/graphical.target por lo que para cambiar de target de arranque, bastará con eliminar dicho enlace y crear uno nuevo apuntando al nuevo target.

También podemos añadir o quitar nuevos servicios a un target determinado, bastará con crear nuevos enlaces simbólicos dentro del directorio *.wants del target de arranque (o de algunos de los que dependa) apuntando a los servicios deseados.

Otra forma de modificar el target de inicio es a través de los parámetros que le pasamos al kernel en el archivo de configuración del gestor de arranque añadiendo por ejemplo systemd.unit=multi-user.target para arrancar en nivel3.

Journal es un componente más de systemd, que capta los mensajes syslog, los mensajes de registro del kernel, los del disco RAM inicial y los mensajes de arranque iniciales, así como mensajes escritos en stdout y stderr de todos los servicios, haciendo que estén disponibles para el usuario. Se puede utilizar en paralelo, o en lugar de un demonio syslog tradicional, como rsyslog o syslog-ng.

Journal es el sustituto de syslog, el cual nos permite filtrar la salida del registro por campos, dejar el registro de forma volátil o volverlo persistente mediante su archivo de configuración (/etc/systemd/journald.conf), darle un tamaño máximo al fichero de registro, reenviar la salida del registro a un terminal, etc.. Para leer el registro se puede utilizar el comando *"**journalctl**"*

Si queremos ver los mensajes del actual inicio del sistema, utilizaremos la opción -b:

journalctl -b

Para añadir un filtro y ver sólo los errores generados en este inicio:

journalctl -b -p err

La opción -p nos permite filtrar los mensajes por su prioridad: "emerg", "alert", "crit", "err", "warning", "notice", "info", "debug".

Muchas veces es necesario revisar los logs generados por un binario concreto. Esto podemos especificarlo de la siguiente manera:

journalctl /usr/bin/<nombre binario>

Para visualizar los últimos registros de forma detallada se puede ejecutar:

journalctl -xe

Para visualizar los registros de un servicio:

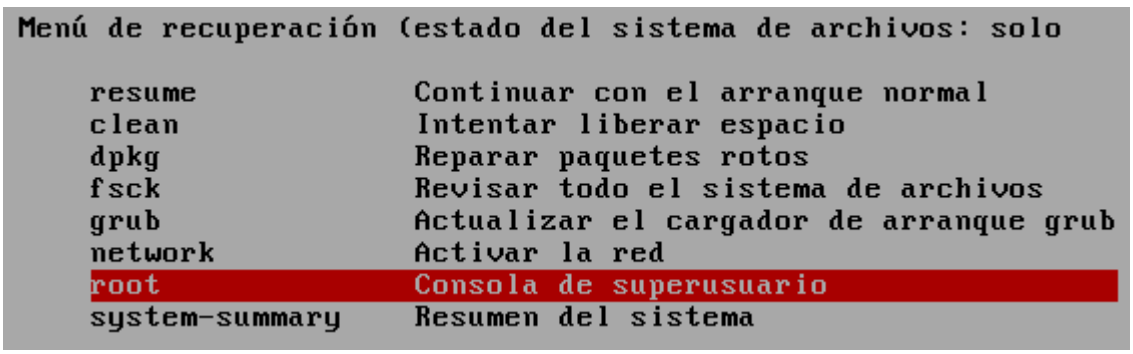
journalctl -u nombre.service

5.4. Modo recuperación

Cuando se instala Ubuntu como único sistema operativo, GRUB hace el arranque del sistema directamente sin mostrar el menú, para acceder al menú de grub, ya sea para entrar al **modo de recuperación** (recovery mode), iniciar el sistema con un kernel anterior, o hacer un chequeo de la memoria RAM con Memtest, se presiona la tecla **Shift** derecha.

Una vez se ha accedido al modo de recuperación tendremos disponibles las siguientes opciones:

- resume: continua con el arranque normal del sistema.
- clean: intenta liberar espacio en disco, esta opción ejecuta automáticamente el comando "*apt autoremove"* buscando paquetes no necesarios.
- dpkg: reparar paquetes rotos, si el problema del sistema es debido a una actualización interrumpida esta opción reparará el sistema de paquetes actualizándolos a la última versión disponible.
- fsck: revisa todo el sistema de archivos reparando los errores encontrados.
- grub: actualiza el gestor de arranque del sistema.
- network: activa la red.
- root: pasa al terminal de comandos con permisos de root, por defecto el sistema de archivos se monta en solo lectura, si vamos a editar algún fichero o ejecutar algún comando que haga cambios en el sistema se puede remontar el sistema de ficheros con permisos de escritura con el comando "*mount -o rw,remount /*"
- system-summary: resumen del sistema.



6. INSTALAR APLICACIONES

Por defecto, muchos programas se instalan en el proceso de instalación de cualquier sistema GNU/Linux. Aún así, es habitual tener que instalar software en particular para alguna tarea no cumplida por el software por defecto. Los principales modos de instalar aplicaciones son:

- Mediante **APT** (Advanced Package Tool):
  - Con aplicaciones gráficas. Como el **centro de software de Ubuntu** o **Synaptic** con las que se puede controlar los paquetes a instalar en el sistema.
  - Mediante comandos **apt, apt-get** o **aptitude**. Estos son programas más avanzados que se ejecutan en modo terminal. **apt-get** fue la primera interfaz, basada en la línea de órdenes, **apt** es un segundo frontend de línea de comandos proporcionado por APT el cual soluciona algunos errores de diseño de la orden apt-get. **aptitude** proporciona una interfaz interactiva en modo consola.
- Mediante paquetes **deb** o **rpm**. Los ficheros con extensión .deb son paquetes de aplicaciones ya preparados para instalarse en sistemas derivados de Debian. Los ficheros con extensión .rpm son paquetes de aplicaciones ya preparados para instalarse en sistemas derivados de Red Hat.
- Mediante paquetes **snap** o **flatpak**. Nuevo sistema de “paquetes independientes”.
- Mediante ficheros **AppImage:** es un formato que surge para distribuir aplicaciones sin necesidad de instalar, y sin necesidad de tener permisos de administrador para su ejecución, independiente de la distribución Linux en la que se ejecute.
- **Archivos binarios**. Suelen ser archivos con extensión .bin, extensión .run o scripts de instalación, normalmente con extensión .sh**.**
- **Archivos fuente.** Es el código de la aplicación que requiere compilación.

6.1. Paquetes y Repositorios

Software es un término muy amplio y se utiliza generalmente para definir los programas que se pueden ejecutar en el ordenador. A menudo cada programa necesita otros recursos para trabajar. Los sistemas GNU/Linux utilizan **paquetes** para almacenar todo lo que un programa en particular necesita para ejecutarse.

Un paquete, entonces, es esencialmente una colección de archivos construidos en un único archivo, el cual puede ser manejado mucho más fácilmente.

Normalmente, cuando alguien empaqueta un programa, incluye el **código fuente** del programa en el paquete. Los **paquetes fuente** son sencillamente paquetes que incluyen código fuente, y generalmente pueden ser utilizados por cualquier tipo de máquina si el código se compila de manera correcta.

Los **paquetes binarios** son los que están construidos específicamente para algún tipo de ordenador o “arquitectura”. Para saber qué arquitectura se está usando, se puede utilizar el comando **arch**.

Los programas a menudo utilizan los mismos archivos que otras aplicaciones. En vez de poner esos archivos en cada paquete, se puede instalar un paquete separado para proporcionar esos archivos a todos los programas que los necesiten. Por eso, al instalar programas que necesitan esos archivos, el paquete que los contiene debe ser instalado. Cuando un paquete depende de otro de esa manera, esto se conoce como “**dependencia de paquete**”.

Un canal de software o **repositorio** es una localización que almacena paquetes de tipos similares, que pueden ser descargados e instalados utilizando el gestor de paquetes. El gestor de paquetes almacena un índice de todos los paquetes disponibles en ese canal de software. A veces se reconstruye este índice para asegurar que todos los datos están al día y qué paquetes han sido actualizados o añadidos al canal desde la última comprobación.

APT almacena una lista de repositorios o canales de software en el fichero ***/etc/apt/sources.list.*** Editando este fichero desde la línea de comandos, podemos añadir, eliminar o deshabilitar temporalmente los repositorios de software.

A partir de Ubuntu 9.10, los repositorios también se puede encontrar en ficheros dentro del directorio **/etc/apt/sources.list.d/** donde se pueden agregar repositorios mediante el comando:

sudo add-apt-repository ppa:[nombre del repositorio]

El repositorio de software de Ubuntu está dividido en cuatro "componentes" en función de la habilidad del equipo de Ubuntu de proporcionar soporte para el software que proporcionan estos componentes y de la comunión con los objetivos de la filosofía de Ubuntu. En base a esto, tenemos 4 componentes:

- Main: contiene aplicaciones que son software libre y con actualizaciones de seguridad y soporte técnico.
- Restricted: software que es soportado por el equipo de Ubuntu a pesar de que no esté disponible bajo una licencia completamente libre.
- Universe: cualquier paquete de software de código abierto y de software disponible bajo una variedad de licencias menos abiertas y sin ninguna garantía de obtener actualizaciones de seguridad y soporte
- Multiverse: software que no es libre.

6.2. APT

El comando **apt** es la herramienta que utiliza Debian y sus derivadas, para gestionar los paquetes disponibles en los repositorios. También disponemos de los comandos **apt-get** y **aptitude**.

Uso de apt:

sudo apt [opciones] orden [paquetes]

Ejemplos de uso del comando "apt":

- Actualizar el listado de paquetes disponibles: ***sudo apt update***
- Instalar los programas deseados: ***sudo apt install paquete***
- Actualizar solo los paquetes ya instalados que no necesitan, como dependencia, la instalación o desinstalación de otros paquetes: ***sudo apt upgrade***
- Actualizar todos los paquetes del sistema, instalando o desinstalando los paquetes que sean necesarios para resolver las dependencias que pueda generar la actualización de algún paquete: ***sudo apt full-upgrade***
- Desinstalar un paquete: ***sudo apt remove paquete***
- Desinstalar un paquete y eliminar los archivos de configuración: ***sudo apt purge paquete***
- Para limpiar los paquetes que ya no se usan: ***sudo apt autoremove***
- Para buscar un paquete determinado: ***sudo apt search paquete***
- Obtener más información de un paquete específico: ***sudo apt show paquete***
- Editar el contenido de nuestros repositorios en el fichero /etc/apt/sources.list con comprobación de errores: ***sudo apt edit-sources***
- Obtener listados de los paquetes instalados: ***sudo apt list --installed***
- Obtener listado de los paquetes actualizables: ***sudo apt list --upgradable***
- Obtener listado de todos los paquetes disponibles: ***sudo apt list --all-versions***

6.3. Paquetes .deb y .rpm

Otra forma de instalar aplicaciones en el sistema es por medio de los paquetes ya preparados para ser instalados y con extensión .deb. Para instalar estos paquetes sólo hay que hacer doble click sobre el fichero y automáticamente se lanzará la aplicación, que se ocupará de instalar el paquete y buscar las dependencias de otros paquetes que pudiera necesitar para su correcta instalación.

Si lo preferimos, también se pueden instalar mediante la línea de comandos, mediante el comando **dpkg**:

sudo dpkg -i <paquete>.deb

En este caso también habrá que instalar manualmente las posibles dependencias del paquete.

El mismo comando también se puede usar para desinstalar el paquete:

sudo dpkg -r <paquete>

Algunas distribuciones de GNU/Linux, como por ejemplo Red Hat, SUSE y Mandriva, usan paquetes **.rpm**, organizados de manera diferente a los paquetes .deb de Debian y Ubuntu. Para instalar estos paquetes es preciso convertirlos antes al formato .deb. Para ello se usa la aplicación **alien.** La aplicación alien se utiliza de la siguiente manera:

sudo alien <paquete>.rpm

6.4. Paquetes universales

Los **paquetes universales** son un tipo de paquete que no sólo incluye el programa, sino que también incluye las dependencias que necesita. De esta manera, sólo tenemos que hacer doble click en el archivo, y se instalará todo lo necesario para que se ejecute correctamente. Dentro de este tipo de paquetes podemos destacar, **snap**, **flatpak** y **appimage**.

Incluir las dependencias tiene como ventajas:

- No tener que instalar más paquetes.
- No tener que usar repositorios adicionales para cumplir las dependencias.
- Podremos instalar las aplicaciones independientemente de la distribución o versión Linux.

El principal punto negativo de los paquetes universales es que sus paquetes ocuparán más tamaño que un paquete normal, al incluir las dependencias.

Otra gran novedad es que todas las apps que instalemos estarán aisladas del resto del sistema, por motivos de seguridad, lo que conlleva que:

- Las dependencias incluidas en los paquetes no afectarán al resto de programas (por ejemplo, dependencias más nuevas o viejas de las que tenemos instaladas).
- Los desarrolladores pueden lanzar actualizaciones más rápidamente, con la seguridad de que si hay un bug no afectarán al sistema de los usuarios y pueden volver a la versión anterior.

Por ejemplo para utilizar paquetes snap tenemos el comando “sudo snap [opciones]”. Algunas de las opciones disponibles son:

- abort: para abortar un cambio pendiente.
- ack: añade una afirmación al sistema.
- changes: muestra los cambios del sistema.
- find: busca paquetes para instalar
- install: instala un snap en el sistema.
- list: muestra una lista de los snaps instalados.
- login: se identifica en la Store.
- logout: sale de la Store.
- refresh: refresca un snap en el sistema.
- remove: elimina un snap del sistema.

Flatpak funciona de forma parecida a snap, mientras que en el empaquetado appimage ni siquiera hay necesidad de realizar la instalación, simplemente es necesario descargar el fichero appimage y ejecutar.

6.5. Archivos binarios

Los archivos binarios no contienen un conjunto de programas o librerías como los paquetes, sino que son el programa en sí mismo. Normalmente se suelen distribuir bajo este sistema programas comerciales, que pueden ser o no gratuitos, pero que normalmente no son libres.

Para instalar este tipo de programas deberemos aplicarle el permiso de ejecución al fichero y ejecutarlo. Para realizar esto desde un terminal:

Damos permisos de ejecución con el comando.

sudo chmod +x [nombre_binario]

Instalamos el archivo binario con el comando:

sudo ./[nombre_binario]

6.6. Compilar programas fuente

A veces se encuentran aplicaciones que no proporcionan paquetes de instalación, y hay que compilar a partir del código fuente. Para ello, lo primero que debemos hacer es instalar un meta-paquete llamado **build-essential**, usando uno de los métodos explicados anteriormente.

En general, los pasos a seguir para compilar una aplicación son los siguientes:

1. Descargar el código fuente.
2. Descomprimir el código, generalmente está empaquetado con tar comprimido bajo gzip (*.tar.gz) o bzip2 (*.tar.bz2).
3. Entrar en el directorio creado al descomprimir el código.

4. Ejecutar el script **./configure** (sirve para comprobar las características del sistema que afectan a la compilación, configurando la compilación según estos valores, y crear el archivo makefile).
5. Ejecutar el comando **make**, encargado de la compilación.
6. Ejecutar el comando **sudo make install**, que instala la aplicación en el sistema. Otra opción mejor es instalar el paquete **checkinstall**, y ejecutar **sudo checkinstall**. Esta aplicación crea un paquete .deb de forma que no haya que compilarlo la próxima vez, aunque no incluye la lista de dependencias. El uso de checkinstall tiene también la ventaja de que el sistema tendrá constancia de los programas instalados de esa forma, facilitando también su desinstalación.

6.7. Aplicaciones de otros sistemas

Se pueden ejecutar aplicaciones creadas para otros sistemas operativos en GNU/Linux utilizando dos métodos diferentes:

- Ejecutando un SO diferente dentro de Linux (a través de la llamada virtualización).
- Utilizando un "intermediario" compatible con Windows, como Wine, Cedega o CrossOver Office.

Generalmente las aplicaciones de escritorio y servicios trabajan mejor con virtualización, por ejemplo utilizando VirtualBox, mientras que las aplicaciones multimedia, como juegos, trabajan mejor con Wine o Cedega ya que estas aplicaciones requieren de requisitos elevados o el uso de aceleración 3D, aún en desarrollo para los sistemas virtualizados.

7. CONTROL DE PROCESOS

7.1. Estado de los procesos

Un proceso es una abstracción que se utiliza en Linux para representar un programa que está en ejecución. Los procesos que se encuentren en ejecución en un determinado momento serán, en general, de diferente naturaleza. Podemos encontrar:

- **Procesos de sistema**, o bien procesos asociados al funcionamiento local de la máquina y del kernel, o bien procesos (denominados **daemons**) asociados al control de diferentes servicios, ya sean locales, o de red, porque estamos ofreciendo el servicio (actuamos de servidor) o estamos recibiendo el servicio (actuamos de clientes). La mayoría de estos procesos aparecerán asociados al usuario root.
- **Procesos del usuario administrador**: en caso de actuar como root, nuestros procesos interactivos o aplicaciones lanzadas también aparecerán como procesos asociados al usuario root.
- **Procesos de usuarios del sistema:** asociados a la ejecución de sus aplicaciones, ya sea tareas interactivas en modo texto o en modo gráfico.

El kernel asigna un número de identificación **PID** a todos los procesos según se van creando. Los procesos en Linux se crean copiando los ya existentes. Cuando se clona un proceso, al original se le conoce como padre y a la copia como hijo. El atributo **PPID** de un proceso hace referencia al PID del proceso padre del que se ha copiado. Siempre habrá un proceso llamado **init**, que hace de padre del resto de procesos del sistema y que se encarga de crear los procesos necesarios durante el inicio del sistema.

A un proceso no se le asigna automáticamente tiempo de ejecución de la CPU por existir. Los procesos pueden estar en el sistema en los siguientes estados:

- **Ejecutable (R):** el proceso puede coger tiempo de CPU.
- **Suspendido (S):** el proceso espera a algún recurso.
- **Suspendido ininterrumpible (D):** el proceso que espera no se puede detener.
- **Zombie (Z):** el proceso intenta finalizar pero se ha quedado “colgado”
- **Detenido (T):** el proceso no tiene permiso para ejecutarse.

7.2. Resumen de comandos para el control de procesos

ps: lista los procesos con sus datos de usuario, tiempo, identificador de proceso y línea de comandos usada. Una de las opciones más utilizada es **ps -ef**, pero hay muchas opciones disponibles como **ps aux** o **ps lax**.

top: una versión que nos da una lista actualizada de procesos a intervalos.

kill: permite eliminar procesos del sistema mediante el envío de señales al proceso como, por ejemplo, la de terminación **kill -9 PID**,donde indicamos el identificador del proceso. Útil para procesos con comportamiento inestable o programas interactivos que han dejado de responder.

sleep: suspende la ejecución durante el número de segundos pasado como argumento.

pstree: muestra la jerarquía de procesos.

nohup: permite que los procesos de un usuario continúen en ejecución aunque se cierre la sesión de este usuario.

7.3. Procesos en primer y segundo plano

Un proceso que corre en **primer plano** (“fg” de foreground) normalmente es aquél que ingresamos por teclado y podemos ver por pantalla. Los procesos tipo demonio no corren en primer plano, sino en segundo, no son llamados desde un terminal (tty) sino desde un puerto, y no aparecen en pantalla. Si queremos pasar un proceso que está corriendo en primer plano a segundo plano, podemos utilizar el símbolo “**&**” después del comando.

Si utilizamos el comando “**jobs**” podremos ver los procesos que están corriendo en segundo plano.

Mientras que “**&**” lleva a segundo plano un proceso existente en primer plano, el comando “**bg**” lo reinicia en segundo plano. “bg” no utiliza el nombre del proceso ni el PID sino el número de trabajo.

Mientras que “bg” es background o segundo plano, “**fg**” es foreground o primer plano. Lo que hace este comando es traer a primer plano un proceso que normalmente corre en segundo plano, utilizando también el número de trabajo.

Para pasar a segundo plano un proceso que corre en primer plano, podemos también utilizar **Ctrl + Z**.

7.4. Prioridad de los procesos

La prioridad o *niceness* de un proceso es un valor numérico que indica al kernel cómo ha de tratar al proceso en relación con el resto de procesos que esperan poder utilizar la CPU. El rango de valores que puede tener **nice** va de -20 a +19, un valor alto implica un nivel de prioridad bajo y un valor bajo implica un nivel de prioridad alto.

El valor **nice** es la prioridad establecida por el usuario y la prioridad **PR** es la prioridad actual del proceso que usa el kernel. La prioridad en el kernel va de 0 a 139, de 0 a 99 para tiempo real y del 100 al 139 para procesos de usuario. La relación entre el valor nice y la prioridad PR es: PR = 20 + NI.

Los procesos que se crean heredan la prioridad de sus padres. El **propietario** de un proceso **puede aumentar** su valor **nice** pero **no bajarlo**. Esta prohibición evita que los procesos de baja prioridad generen procesos hijo de alta prioridad. El superusuario si que puede asignar a nice cualquier valor.

nice: puede determinar la prioridad de un proceso durante su creación.

Sintaxis: nice –n [prioridad] [comando]

renice: permite modificar la prioridad de un proceso que está en ejecución.

Sintaxis: renice [prioridad] [PID] [opciones]

Algunas opciones:

- -g [grupo]: cambia la prioridad de los procesos de un grupo
- -u [usuario]: cambia la prioridad de los procesos de un usuario

7.5. Planificación de procesos

En Linux, el responsable de realizar todas las tareas periódicas es el **demonio cron** que se inicia con el sistema y sigue activo hasta que el sistema se apague. El cron lee uno o varios archivos de configuración, en los cuales hay una serie de líneas de comandos y horas que indican cuándo se han de ejecutar.

El archivo de configuración de cron es **crontab** que suele estar en varios directorios como */var/spool/cron/crontabs, /etc/cron.d* y */etc/crontab.* Cuando cron se inicia lee todos los archivos de configuración, los guarda en memoria y luego pasa a un estado de suspensión. Cada minuto, cron se activa, comprueba las horas de modificación de los archivos crontab, vuelve a cargarlos en memoria y ejecuta las tareas programadas para ese minuto.

at: permite la ejecución diferida de órdenes en el momento especificado. Estos procesos se ejecutan con el shell Bourne sh.

Sintaxis: at [opciones] [hora fecha + incremento]
órdenes o programas
….
[Ctrl + d]

La "hora" (hasta cuatro dígitos) especifica hora y minutos; puede sustituirse por algunos términos especiales, como:

- now:(ahora)
- noon: (mediodia)
- midnight: (medianoche)

El formato de hora de la orden at permite separar horas y minutos por dos puntos. Si el argumento "hora" tiene 1 ó 2 dígitos, se interpretan como horas. Para indicar solo minutos hay que precederlos por ceros. Se supone un reloj de 24 horas (salvo que se especifique am o pm).

La "fecha" es opcional y admite los siguientes formatos:

- Mes y día: (May 13)
- Mes, día y año: (Aug 23 2008)
- Día de la semana: (Monday)
- Día especial: (Today, Tomorrow)

El formato de fecha admite abreviaturas de tres letras. El incremento es optativo y consiste en el signo + seguido (tras un espacio) de un número y uno de los siguientes términos: minutes, hours, days, weeks, months, years.

Algunas opciones:

- -l: muestra la lista de identificadores asociados a los trabajos diferidos de un usuario o de todos si lo ejecuta root.
- -r nn: elimina de la cola de trabajos diferidos los correspondientes a los identificadores indicados. Si es root el que lo ejecuta puede poner * y elimina todos.

batch: permite la ejecución diferida de órdenes cuando la "carga" del sistema lo permita. Las características de UNIX como sistema multitarea y multiusuario hace que en algún momento el número de procesos en curso pueda ser muy elevado. Esto es especialmente crítico en los sistemas monoprocesador. El método de comparación de tiempo por el procesador puede llevar a una sobrecarga. En estos casos es cuando se aprecia el efecto de la orden batch.

Sintaxis: batch
órdenes o programas
[Ctrl + d]

También puede ejecutarse redirigiendo la entrada de un archivo que contenga las órdenes o programas.

Ejemplo: batch < archtrabajo

crontab: crea los archivos con las especificaciones de ejecución periódica de órdenes que utiliza cron. La orden crontab se usa para crear órdenes o archivos que van a ser gestionados por el proceso cron en un tiempo determinado.

Sintaxis: crontab arch-esp

El archivo de especificaciones arch-esp constará de una línea por cada tarea que se vaya a repetir periódicamente, con seis entradas separadas por espacios. Los seis campos son los siguientes:

- 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 0 a 6) (0 = domingo)
- Orden o programa a ejecutar

El signo * significa "todos" (cualquier valor del rango). En los archivos crontab puede incluirse el signo % en el último campo (el sexto) para indicar una entrada estándar. También pueden incluirse varios valores en uno de los 5 primeros campos separándolos por comas. Por ejemplo, 00,15,30,45 8-10 * * * significa cada 15 minutos entre las 8 y las 10 horas de todos los días.

Ejemplos:

- 0 12 * * * Todos los días a las 12 en punto.
- 30 8 * * 1 Todos los lunes a las 8.30.
- * 14 31 7 * Cada minuto desde las 14 a las 15 horas del 31 de julio de cada año.
- 0 12 15 * 5 Los viernes y los días 15 de cada mes a las 12.00.
- 0 12 15 * * echo “Mediodía del día 15”: mostrará el mensaje "Mediodía del día 15" a las 12 horas, 0 minutos del día 15 de cada mes.

Algunas opciones:

- -l Lista el archivo crontab actual.
- -r Elimina el crontab actual.
- -e Edita el crontab actual.

Cada usuario sólo puede poseer un archivo crontab. La orden **crontab archivo-mio** establece, como archivo crontab el especificado (archivo-mio). Pueden elaborarse varios archivos para ejecución de acciones periódicas pero sólo tendrán efecto las contenidas en el último archivo de la orden crontab.

Es habitual utilizar crontab para realizar periódicamente operaciones de mantenimiento de los archivos propios. El uso y gestión de la orden crontab, que actualiza el archivo, están reservados para los usuarios que figuran en /usr/lib/cron/cron.allow y es denegado para los usuarios que se encuentran en /usr/lib/cron/cron.deny. Todos los procesos que ejecuta la orden cron se registran en el archivo /usr/lib/cron/log si la variable CRONLOG está activa.

watch: permite la ejecución de un comando cada cierto tiempo, si no se especifica nada el tiempo por defecto son 2 segundos.

Sintaxis: watch [-n tiempo] [comando]

8. GESTIÓN DE USUARIOS Y GRUPOS

8.1. La cuenta del administrador

Por razones de seguridad, las tareas administrativas en Ubuntu sólo pueden ser realizadas por usuarios con privilegios especiales. Un usuario normal puede realizar todas las tareas habituales que se necesitan a diario, pero las tareas de administración del sistema solo podrán realizarlas usuarios autorizados.

En GNU/Linux (y Unix en general) existe un superusuario llamado **root**. Este usuario tiene acceso total al sistema, pudiendo instalar, modificar y eliminar cualquier archivo o programa, trabajar con este usuario es potencialmente peligroso, por lo que solo en recomendable para gente experimentada, que sepa lo que hace. Además, solo debería usarse para tareas de administración del sistema, nunca para realizar las actividades habituales de un usuario normal.

Por defecto, Ubuntu tiene la cuenta de root parcialmente deshabilitada. No podemos loguearnos como root para entrar en el sistema puesto que no tiene definida ninguna contraseña de acceso. Para poder loguearnos como root la primera vez deberemos asignarle un password. Esto podemos hacerlo desde el terminal:

Primero nos hacemos root:

sudo su

Y después creamos la contraseña:

sudo passwd

La primera cuenta de usuario que se crea en el sistema durante la instalación, por defecto, está capacitada **para usar sudo** y realizar las tareas administrativas necesarias. Además, podemos restringir o permitir el acceso a sudo al resto de usuarios del sistema, haciéndolos miembros del **grupo sudo**.

Sudo permite implementar un control de acceso altamente granulado sobre que usuarios ejecutan que comandos. Si un usuario normal desea ejecutar un comando de root, o de cualquier otro usuario, sudo verifica en su lista de permisos y sudo sólo lanza su ejecución si realmente está permitido. Es decir, sudo es un programa que basado en una lista de control, almacenada en **/etc/sudoers**, permite o no permite la ejecución al usuario que lo invocó sobre un determinado programa propiedad de otro usuario, generalmente del administrador del sistema o root.

Los usuarios deben confirmar su identidad al ejecutar sudo, proporcionando su propia contraseña antes de ejecutar el programa requerido. Una vez que se ha autenticado el usuario, y si el archivo de configuración /etc/sudoers permite dar acceso al usuario al comando requerido, el sistema lo ejecuta y lo registra.

Por defecto, después de invocar a sudo se dispone de 5 minutos para reutilizar el mismo u otros comandos sobre los que se tenga derecho, sin necesidad de ingresar la contraseña de nuevo. Si se necesita extender el tiempo por otros 5 minutos se utiliza la opción **sudo –v**, de validate. Por el contrario, si se han terminado las tareas implicadas, se puede usar **sudo –k**, de Kill, para concluir con el tiempo de gracia de la validación.

Para saber los comandos que puede usar un usuario mediante este comando se utiliza la opción **sudo –l.** También pueden emplearse comandos de otros usuarios del sistema indicando la opción **–u**.

Una de las opciones más interesantes es la que permite editar archivos de texto de root, como se verá más adelante con el permiso otorgado en sudoers. Para ello se emplea la opción **–e**, que está ligada a otro comando de sudo llamado **sudoedit** que invoca al editor por defecto del usuario, y que generalmente es vi.

Cuando se configura sudo se dispone de múltiples opciones que se pueden establecer, y que podemos consultar con la opción **–L**, mostrándonos las opciones y una breve descripción. Estás opciones se establecen en el archivo de configuración **/etc/sudoers**.

El comando **visudo** permite la edición del archivo de configuración de sudo denominado sudores, invocando al editor por defecto que suele ser vi. El comando visudo cuando es usado, bloquea el archivo /etc/sudoers, de tal manera que nadie más lo puede utilizar.

Este bloqueo se produce por razones obvias de seguridad, para evitar que varios administradores modifiquen accidentalmente los cambios de otros administradores.

Otra característica importante de visudo es que al cerrar el archivo, verifica esté bien configurado. Es decir, detectará si hay errores de sintaxis en sus múltiples opciones o reglas de acceso que se tengan. Por esta razón, aunque es posible, no debe editarse /etc/sudoers directamente, y debe editarse a través de visudo.

Si sólo se desea comprobar que /etc/sudoers está bien configurado, se usa la opción **–c** que, salvo que se indique un archivo de configuración alternativo, toma la configuración por defecto.

La opción **–s** activa el modo 'estricto' del uso de visudo, por lo que no solo se comprobará a nivel sintáctico sino también el orden correcto de las reglas. Por ejemplo, si se define el alias para un grupo de comandos y se usa antes de su definición, se detectará con esta opción.

Es más fácil entender **sudoers** si dividimos su configuración en tres partes:

- Alias.
- Opciones (Defaults).
- Reglas de acceso.

Por extraño que parezca, ninguna de las secciones es obligatoria, ni tiene un orden específico, aunque no tiene sentido carecer de la tercera, que contiene la definición de los controles o reglas de acceso.

Un **alias** se refiere a un usuario, un comando o a un equipo. El alias engloba bajo un solo nombre, denominado nombre del alias, una serie de elementos a los que posteriormente, en la parte de definición de reglas, serán referidos mediante ese único nombre. Las formas para crear un alias son las siguientes:

tipo_alias NOMBRE_DEL_ALIAS = elemento1, elemento2, elemento3, ... elementoN

tipo_alias NOMBRE1 = elemento1, elemento2 : NOMBRE2 = elemento1, elemento2

En el segundo caso, separado por ":" es posible indicar más de un alias en una misma definición.

El tipo_alias define los elementos, que dependiendo del tipo de alias serán sus elementos. Los tipos de alias son cuatro:

- Cmnd_Alias: define alias de comandos.
- User_Alias: define alias de usuarios normales.
- Runas_Alias: define alias de usuarios con los que podremos identificarnos al ejecutar un comando.
- Host_Alias: define alias de hosts o equipos.

El nombre del alias puede llevar letras, números o guión bajo (_), y debe comenzar con una letra mayúscula. Se acostumbra a usarlos siempre en mayúsculas. Los elementos del alias varían dependiendo del tipo de alias. A continuación se detallan los tipos de alias y sus elementos.

Cmnd_Alias define uno o más comandos y otros alias de comandos que podrán ser utilizados después. El siguiente ejemplo indica que a quien se le aplique el alias WEB podrá ejecutar los comandos apachectl, httpd y editar todo lo que este debajo del directorio /etc/httpd/. Nótese que cuando se indican directorios deben terminar con '/'. Debe indicarse la ruta completa a los comandos.

```
Cmnd_Alias WEB = /usr/sbin/apachectl, /usr/sbin/httpd, sudoedit /etc/httpd/
```

En el ejemplo que sigue, al usuario que se le asigne el alias APAGAR podrá hacer uso del comando 'shutdown' con los parámetros tal y como están indicados:

```
Cmnd_Alias APAGAR = /usr/bin/shutdown -h 23\:00
```

Nótese que es necesario escapar el signo ':', así como los símbolos ' , = ó \.

NET_ADMIN es un alias con los comandos de configuración de interfaces de red ifconfig y de firewall iptables, pero además le agregamos un alias previamente definido que es WEB, así que a quien se le asigne este alias podrá hacer uso de los comandos del alias WEB.

```
Cmnd_Alias NET_ADMIN = /sbin/ifconfig, /sbin/iptables, WEB
```

A quien se le asigne el siguiente alias podrá ejecutar todos los comandos que estén dentro del directorio /usr/bin/, a excepción del comando 'rpm' ubicado en el mismo directorio.

Cmnd_Alias TODO_BIN = /usr/bin/, !/usr/bin/rpm

NOTA IMPORTANTE: este tipo de alias con unos permisos muy amplios menos '!' algo, generalmente no son una buena idea, ya que comandos nuevos que después se añadan a ese directorio también podrán ser ejecutados. Siempre es más seguro definir específicamente lo que se requiera.

User_Alias definen a uno o más usuarios, grupos del sistema (indicados con %), grupos de red (netgroups indicados con +) u otros alias de usuarios. El siguiente ejemplo indica que al alias MYSQL_USERS pertenecen los usuarios indicados individualmente más los usuarios que formen parte del grupo 'mysql'.

User_Alias MYSQL_USERS = andy, marce, juan, %mysql

En el siguiente, 'raul' y 'ana' pertenecen al alias ADMIN.

User_Alias ADMIN = raul, ana

Aquí encontramos algo nuevo, definimos el alias de usuario TODOS, que al poner como elemento la palabra reservada 'ALL' abarcaría a todos los usuarios del sistema. Sin embargo, no deseamos a dos de ellos, por lo que negamos con admiración ('!'), que serían los usuarios 'samuel' y 'david'. Es decir, todos los usuarios menos esos dos.

User_Alias TODOS = ALL, !samuel, !david

Este tipo de alias tan amplios no es recomendable.

Por último, en este ejemplo se incluyen los usuarios de ADMIN junto con el usuario 'alejandra'.

User_Alias OPERADORES = ADMIN, alejandra

Runas_Alias funciona exactamente igual que User_Alias, la única diferencia es que es posible usar el UID de un usuario con el carácter '#', y sirve para especificar usuarios de los que podremos tomar las credenciales al ejecutar los comandos asociados.

En el siguiente ejemplo, al alias OPERADORES pertenecen el usuario con UID 501 y el usuario 'fabian':

Runas_Alias OPERADORES = #501, fabian

Host_Alias definen uno o más equipos u otros alias de host. Los equipos pueden indicarse por su nombre (si se encuentra en /etc/hosts), por nombre de dominio (si existe un resolvedor de dominios), por dirección IP, o por IP con máscara de red.

El alias LANS define todos los equipos de las redes locales:

Host_Alias LANS = 192.168.0.0/24, 192.168.0.1/255.255.255.0

Se define dos alias en la misma línea WEBSERVERS y DBSERVERS con sus respectivas listas de elementos, el separador ':' es válido en cualquier definición de tipo de alias:

Host_Alias WEBSERVERS = 172.16.0.21, web1 : DBSERVERS = 192.168.100.10, dataserver

Las **opciones por defecto**, o **defaults**, permiten definir ciertas características de comportamiento para los alias previamente creados, para usuarios, usuarios privilegiados, equipos o de manera global para todos. No es necesario definir opciones por defecto, ya que las tiene previamente establecidas sudo. Sin embargo, la potencia de sudo está en su alta granularidad de configuración, así que es importante conocer como establecer opciones específicas.

Es posible establecer las opciones en cuatro niveles de uso:

- Global
- Por usuario.
- Por usuario privilegiado.
- Por equipo (host).

Se usa la palabra reservada 'Defaults' para establecer las opciones y, dependiendo del nivel al que deseamos afectar, su sintaxis es:

Global: *Defaults opcion1, opcion2 ...*
Usuario: *Defaults:usuario opcion1, opcion2 ...*
Usuario Privilegiado: *Defaults>usuario opcion1, opcion2 ...*
Equipo: *Defaults@equipo opcion1, opcion2 ...*

La lista de opciones es algo extensa, y puede consultarse en las páginas del manual, con “man sudores”. A continuación vamos a ver varios ejemplos del uso de establecer opciones.

Los defaults los divide el manual (man sudoers) en cuatro: flags o booleanos, enteros, cadenas y listas. Veamos entonces algunos ejemplos de uso para cada uno de ellos:

Flags o **booleanos**, generalmente se usan de manera global. Simplemente se indica la opción y se establece a 'on', o para desactivarla 'off' se antepone el símbolo '!' a la opción. Es necesario consultar el manual para saber el valor por defecto 'on' u 'off', y poder saber si realmente necesitamos invocarla o no. El siguiente ejemplo establece a 'on' la opción 'mail_always' que enviará un correo avisando cada vez que un usuario utiliza sudo. Esta opción requiere, a su vez, que 'mailto_user' este establecida:

Defaults mail_always

Desactiva 'off' el default 'authenticate', por defecto activo, que establece que todos los usuarios que usen sudo deben identificarse con su contraseña. Obviamente sólo se trata de un ejemplo, y sería una pésima idea usarlo realmente, ya que ningún usuario necesitaría autenticarse. La segunda opción 'log_host', por defecto en 'off', la activamos y bitacoriza el nombre del host cuando se usa un archivo, en vez de syslog, como bitácora de sudo.

Defaults !authenticate, log_host

Aquí se aprecia algo más lógico, usamos opciones por usuario en vez de global, indicando que el usuario 'ana' no requerirá autenticarse. Pero todos los demás si.

Defaults:ana !authenticate

Opciones para usuarios privilegiados, en vez de usar una lista de usuarios, usamos un alias 'ADMIN' que se supone fue previamente definido, y establecemos a 'on' la opción 'rootpw' que indica a sudo que los usuarios en el alias 'ADMIN' deberán usar la contraseña de 'root' en vez de la propia.

Defaults>ADMIN rootpw

Enteros, tal como su nombre lo indica, manejan valores de números enteros en sus opciones, que deben entonces usarse como “opción = valor”. Ejemplo donde se aprecia el uso de opciones con valores enteros. En este caso se establecen opciones para los usuarios 'fernanda' y 'regina' solamente, que solo tendrán una oportunidad de ingresar la contraseña correcta 'passwd_tries' el valor por defecto es de 3 y tendrán un minuto para ingresarla 'passwd_timeout' el valor por defecto son 5 minutos:

Defaults:fernanda, regina passwd_tries = 1, passwd_timeout = 1

La mayoría de las opciones de tiempo o de intentos, al establecerlas con un valor igual a cero entonces queda ilimitada la opción.

En el siguiente ejemplo, se establecen opciones solo para los usuarios que se conectan al servidor 'webserver' y el valor 'umask' indica que si mediante la ejecución del comando que se invoque por sudo es necesario crear archivos o directorios, a estos se les aplicará la máscara de permisos indicada en el valor de la opción.

Defaults@webserver umask = 011

Cadenas, son valores de opciones que indican mensajes, rutas de archivos, etc. Si hubiera espacios en el valor es necesario encerrar el valor entre comillas dobles (" "). En el siguiente ejemplo, para todos los usuarios, cuando se equivoquen al ingresar la contraseña se cambiar el mensaje de error, que por defecto es "Sorry: try again".

Defaults badpass_message = "Intenta de nuevo: "

Listas, permite establecer/eliminar variables de entorno propias de sudo. Los 'Defaults' para variables es de los menos usados en las configuraciones de sudo y ciertamente de los más confusos. Solo existen tres opciones de listas: env_check, env_delete, env_reset y env_keep. Las listas pueden ser remplazadas con '=', añadidas con '+=', eliminadas con '-=' o deshabilitadas con '!'. El siguiente ejemplo, elimina la variable de entorno 'HOSTNAME', pero preserva todas las demás que hubiera. Los comandos que se ejecuten bajo sudo y que requieran de esta variable no la tendrían disponible:

Defaults env_delete -= HOSTNAME

En el siguiente, la primera opción 'env_reset' reinicializa las variables de entorno que sudo utilizará o tendrá disponibles, y sólo quedan disponibles LOGNAME, SHELL, USER y USERNAME. La siguiente línea indica que agregue (+=) a lo anterior, también la variable de entorno DISPLAY a su valor establecido antes del reset.

Defaults env_reset
Defaults env_check += DISPLAY, PS1

Reglas de Acceso, aunque no es obligatorio declarar alias, ni opciones (defaults), y de hecho tampoco reglas de acceso, sin éstas el archivo /etc/sudoers no tendría ninguna razón de ser. Las reglas de acceso definen que usuarios ejecutan que comandos bajo que usuario y en que equipos.

La mejor manera de entender y aprender a configurar sudoers es con ejemplos, a partir de su sintaxis básica:

usuario host = (usuario objetivo) comando1, comando2, ... comandoN

De esta sintaxis podemos extraer que:

- “usuario” puede ser un usuario, un alias de usuario o un grupo (indicado por %),
- “host” puede ser ALL o cualquier equipo, un solo equipo, un alias de equipo, una dirección IP o una definición de red IP/máscara,
- “usuario objetivo” determina como qué usuario podremos ejecutar dicho comando
- “comandos” es cualquier comando indicado con su ruta completa. Si se termina en “/” como en /etc/http/ entonces indica todos los archivos dentro de ese directorio.

En el siguiente caso, el usuario 'daniela' puede utilizar iptables en cualquier host o equipo:

daniela ALL = /sbin/iptables

A continuación, los usuarios definidos en el alias 'ADMIN' desde cualquier host pueden ejecutar cualquier comando.

ADMIN ALL = ALL

En el ejemplo que sigue, los usuarios que pertenezcan al grupo del sistema llamado 'gerentes' pueden ejecutar como si fuera el usuario 'director' la aplicación llamada 'facturacion' en el equipo llamado 'dbserver'. Además, pueden ver el contenido de los archivos que contenga el directorio /var/log como usuarios administradores.

*%gerentes dbserver = (director) /usr/facturacion, (root) /var/log/**

El caso anterior introduce algo nuevo, que en la lista de comandos es posible indicar bajo que usuario se debe ejecutar el permiso. Por defecto es el usuario root, pero no siempre tiene que ser así. Además la lista “hereda” la primera definición de usuario que se indica entre paréntesis ( ), por eso si se tiene más de uno hay que cambiar de usuario en el comando conveniente.

El ejemplo anterior también sería válido de la siguiente manera, puesto que no es necesario indicar “(root)”, ya que es el usuario bajo el cual se ejecutan los comandos por defecto. También es válido usar (ALL) para indicar bajo cualquier usuario:

%gerentes dbserver = /var/log/, (director) /usr/facturacion*

El ejemplo siguiente da permisos absolutos. Para ello se establece permiso para que en cualquier host el usuario 'sergio' pueda ejecutar cualquier comando de cualquier usuario, por supuesto incluyendo los de root.

sergio ALL = (ALL) ALL

A continuación tenemos una regla formada sólo por alias. En el alias de usuario 'SUPERVISORES' los usuarios que estén indicados en ese alias, tendrán permiso en los equipos definidos en el alias de host 'PRODUCCION', de ejecutar los comandos definidos o listados en el alias de comandos 'OPERACION'.

SUPERVISORES PRODUCCION = OPERACION

En este último ejemplo se aprecia lo útil que pueden ser los alias, ya que una vez definida la regla, solo debemos agregar o eliminar elementos de las listas de alias definidos previamente. Es decir, se agrega un equipo más a la red, se añade al alias 'PRODUCCION', un usuario renuncia a la empresa, alteramos el alias 'SUPERVISORES' eliminándolo de la lista, etc.

El siguiente es un ejemplo muy interesante de potencia y flexibilidad. Al usuario 'checo', desde cualquier equipo, tiene permiso de cambiar la contraseña de cualquier usuario (usando el comando 'passwd'), excepto '!' la contraseña del usuario 'root'. Lo anterior se logra mediante el uso de argumentos en los comandos. En el primer ejemplo '/usr/bin/passwd *' el asterisco indica una expansión de comodín (wildcard) que indica cualquier argumento, es decir, cualquier usuario. En el segundo caso '!/usr/bin/passwd root', si indica un argumento específico 'root', y la '!' como ya se sabe indica negación, negando entonces el permiso a cambiar la contraseña de root.

*checo ALL = /usr/bin/passwd *, !/usr/bin/passwd root*

Cuando se indica el comando sin argumentos “/sbin/iptables”, sudo lo interpreta como 'puede usar iptables con cualquiera de sus argumentos'.
En el siguiente ejemplo, al estar un comando entre comillas dobles, sudo lo interpreta como 'puede hacer uso del comando lsmod pero sin argumentos'. En este caso el usuario 'mariajose' podrá ver la lista de módulos del kernel, pero sólo eso.

mariajose ALL = "/sbin/lsmod"

Cuando se definen reglas, en la lista de comandos, estos pueden tener cero (como en los ejemplos anteriores) o más tags. Existen 6 de estas etiquetas o tags, emparejadas dos a dos.

- **PASSWD** y **NOPASSWD**. Por defecto sudo requiere que cualquier usuario se identifique o autentifique con su contraseña. Es posible indicar que un usuario o alias de usuario no requiera de dicha autenticación. Pero el control granular propio de sudo, permite ir aun más lejos al indicar a nivel de comandos, cuáles requieren contraseña para su uso y cuáles no. En el siguiente ejemplo, el usuario 'gerardo' en el equipo 'webserver' no requerirá contraseña para los comandos listados. El tag se hereda, por lo que no sólo afecta al primer elemento de la lista de comandos, sino también a los siguientes.

  gerardo webserver = NOPASSWD: /bin/kill, /usr/bin/lprm, /etc/httpd/conf/

- **EXEC** y **NOEXEC**. Este es un tag muy importante a considerar cuando se otorgan permisos sobre programas que permiten salidas a shell, o shell escape, como es el caso del editor vi, que mediante el uso de '!' permite ejecutar un comando en el shell. Con el tag NOEXEC se logra que esto no suceda, aunque no hay que tomarlo como un hecho, ya que siempre existe la posibilidad de vulnerabilidades no conocidas en los múltiples programas que utilizan escapes a shell. Al igual que los tags anteriores, el tag se hereda y se deshabilita con su tag contrario (EXEC), en caso de que en la lista de comandos hubiera varios comandos. Un ejemplo de uso sería:

  valeria ALL = NOEXEC: /usr/bin/vi

- **SETENV** y **NO SETENV**. Una de las múltiples opciones que pueden establecerse en la sección 'Defaults' u 'opciones' es la opción booleana o de flag 'setenv' que, por defecto y para todos los usuarios, está establecida en 'off'. Esta opción si se activa por usuario (Defaults:sergio setenv) permitirá al usuario indicado cambiar el entorno de variables del usuario del cual tiene permisos de ejecutar comandos, y como generalmente este es 'root' resulta bastante peligrosa. A nivel de lista de comandos, es posible entonces especificar el tag 'SETENV' a un solo comando o a una pequeña lista de éstos y, de esta forma, sólo cuando se ejecuten estos se podrán alterar su entorno de variables. Es decir, en vez de establecerlo por

usuario, sería más conveniente establecerlo por comando a ejecutarse solamente. En el siguiente ejemplo, los usuarios definidos en el alias de usuario 'ADMIN' en cualquier host pueden alterar las variables de entorno cuando ejecuten el comando 'date', que puede ser útil por ejemplo para cambiar variables del tipo LOCALE. Sin embargo, no tendrá esta opción en ninguna otra aplicación.

ADMIN ALL = SETENV: /bin/date, NOSETENV ALL

Otra solución equivalente podría ser:

ADMIN ALL = ALL, SETENV: /bin/date

8.2. Gestión de usuarios y grupos locales

Las dos maneras más habituales de gestionar los usuarios y los grupos locales en GNU/Linux son:

- Gráfica: Sistema --> Administración --> Usuarios y Grupos (es posible obtener una ventana de administración más avanzada *"sudo apt install gnome-system-tools"*)
- Línea de comandos

Para añadir usuarios y grupos al sistema se emplean los scripts **adduser** y **addgroup**. La operación de estos comandos se configura en el fichero ***/etc/adduser.conf***. Por ejemplo, para crear un usuario llamado raul:

adduser raul

Al ejecutar el comando adduser, el sistema pedirá alguna información adicional sobre el usuario y un password o clave. Por defecto, se crea un grupo con el nombre del usuario y éste será el **grupo principal** del usuario por defecto.

Siempre que se añada un usuario al sistema se creará un grupo con su mismo nombre, llamado **grupo primario**. Durante la creación o posteriormente, se podrá incorporar el usuario a otros **grupos secundarios**.

Para añadir un usuario al sistema estableciendo otro grupo como su grupo principal:

adduser --ingroup users raul

Para añadir un usuario (previamente creado) a un grupo:

adduser raul admin

Cuando el número de usuarios es numeroso y heterogéneo, puede ser necesario añadir nuevos grupos. Esto se hace con el comando **addgroup**. Por ejemplo:

addgroup alumnos

Alternativamente a los comandos anteriores, se pueden añadir usuarios y grupos empleando **useradd** y **groupadd**. Estos comandos leen información de configuración del fichero **/etc/login.defs**.

Para eliminar usuarios y grupos se emplean **userdel** y **groupdel** respectivamente. Por ejemplo: "*userdel raul"* elimina el usuario raul. Si además se indica la **opción -r**, también se borrará el directorio personal del usuario con todo su contenido, por defecto /home/raul.

Para eliminar grupos se puede usar el comando **groupdel:**

groupdel alumnos

Para modificar las características de los usuarios y grupos se emplean los comandos **usermod** y **groupmod**. Algunos ejemplos:

usermod -d /home/profes/raul -m

El ejemplo anterior cambia el directorio de inicio del usuario raul para que sea /home/profes/raul. La opción -m hace que mueva el contenido del antiguo directorio al nuevo emplazamiento.

Para cambiar el grupo inicial del usuario raul para que sea profes.

usermod -g profes raul

Para cambiar el nombre del grupo profes a profesores.

groupmod -n profesores profes

Para eliminar un usuario de un grupo.

deluser raul profes

Algunos **ficheros relacionados** con las cuentas de usuario son:

- /etc/passwd: contiene información sobre cada usuario: ID, grupo principal, descripción, directorio de inicio, shell, etc.
- /etc/shadow: contiene los passwords encriptados de los usuarios.
- /etc/group: contiene los miembros de cada grupo, excepto para el grupo principal, que aparece en /etc/passwd.
- /etc/skel: directorio que contiene el contenido del directorio de los nuevos usuarios.

Tanto los usuarios como los grupos se identifican por el sistema a través de un identificador (ID) numérico. El usuario root siempre tiene el ID cero. Cada usuario cuando se conecta al sistema posee un identificador de usuario asociado (**UID**) y un identificador de grupo (**GID**). Para comprobar el uid de nuestro usuario, y el gid del grupo principal al que pertenece, se puede ejecutar la orden **id**.

8.3. Gestión de las contraseñas

Una política de contraseñas rigurosa es uno de los aspectos más importantes para mantener la seguridad del sistema. Muchas violaciones exitosas de seguridad han sido posibles simplemente mediante ataques de fuerza bruta y diccionarios frente a contraseñas débiles. Si se desea ofrecer alguna forma de acceso remoto al sistema local de contraseñas, es recomendable comprobar que se cumplen los requisitos mínimos de complejidad de las contraseñas y los tiempos de vida máximo de las contraseñas, y llevar a cabo auditorías frecuentes de los sistemas de autenticación.

Se recomienda forzar a los usuarios a cambiar sus contraseñas de forma periódica (p.ej. 60 días) y que no puedan ser cambiadas más de una vez en un día (p.ej. en caso de que tengamos configurado el sistema para mantener un histórico de contraseñas irrepetibles, el usuario no podrá cambiar su contraseña N veces en el mismo día hasta volver a tener la misma). Para ello editamos '**/etc/login.defs**' y establecemos:

```
# Maximum number of days a password is valid.
PASS_MAX_DAYS   60
# Minimum number of days before a user can change the password since the last change.
PASS_MIN_DAYS   1
#Number of days when the password change reminder starts.
PASS_WARN_AGE   15
```

Adicionalmente, si un usuario no cambia su contraseña 2 semanas después de haber caducado, se recomienda bloquear la cuenta. De esta forma el sistema desactivará las cuentas de usuario que no estén siendo utilizadas y minimizaremos el riesgo de accesos indebidos. Debemos editar '**/etc/default/useradd**':

```
# Number of days after password expiration that account is disabled.
INACTIVE=14
```

Para ver de manera sencilla el estatus actual de una cuenta de usuario use la siguiente sintaxis:

```
sudo chage -l nombre de usuario
```

Para establecer valores de gestión de contraseñas a un usuario también se puede usar el comando chage, respondiendo a las preguntas interactivas:

sudo chage nombre de usuario

El siguiente también es un ejemplo de cómo se puede cambiar manualmente la fecha explícita de expiración (-E) al 31/01/2011, la duración mínima de la contraseña (-m) a 5 días, la duración máxima de la contraseña (-M) a 90 días, el periodo de inactividad (-I) a 5 días tras la expiración de la contraseña, y el periodo de tiempo de advertencia (-W) a 14 días antes de la expiración de la contraseña.

sudo chage -E 01/31/2011 -m 5 -M 90 -I 30 -W 14 nombre de usuario

8.4. Perfiles de usuario

Para cualquier usuario dado de alta en el sistema se creará un directorio dentro del directorio **/home** con el nombre del usuario que contendrá el perfil que se le aplicará cuando inicie sesión en Linux. El usuario será el único que tendrá todos los derechos de uso. En este perfil se copiará, por defecto, todo el contenido del directorio **/etc/skel.**

Por seguridad el usuario root tiene su propio perfil local de usuario ubicado en el directorio **/root**. Se pueden crear scripts y ficheros de inicio que se ejecutarán al entrar en sesión un usuario de manera que podamos configurar el perfil de trabajo del usuario dentro del sistema.

Existe el fichero **/etc/profile** que contiene el perfil igual para todos los usuarios, en su interior podemos poner comandos que se ejecutarán al iniciar sesión cualquier usuario, también ejecuta todos los script que se encuentran en el directorio **/etc/profile.d/**.

Para ejecutar scripts en el arranque del sistema como el usuario **root**, y que afecten a todos los usuarios, podemos ponerlos en el fichero **/etc/rc.local** o crear un servicio o daemon con Systemd. Para ello debemos crear un fichero dentro de **/etc/systemd/system** con extensión .service que contendrá la configuración del proceso que queremos ejecutar, por ejemplo:

```
[Unit]
Description=Ejemplo de servicio que se ejecuta en el arranque
After=network.target

[Service]
User=root
ExecStart=/opt/scripts/ejemplo.sh

[Install]
WantedBy=multi-user.target
```

Sin entrar en los detalles técnicos que forman este fichero, indicar lo siguiente:

- Description: Podemos introducir aquí el nombre y una descripción de lo que va a hacer nuestro daemon.
- After: Indicaremos si queremos que se cargue después de otros servicios o componentes del sistema. También podemos encontrar una directiva “Before” que actúa al contrario.
- User: El usuario que ejecutará el daemon. También puede funcionar con Group.
- ExecStart: Ruta completa (no funcionan relativas) al script o binario que queremos ejecutar.
- WantedBy: Directivas de uso y otras dependencias.

Una vez que hemos creado el servicio, o daemon, en nuestro sistema, antes de poder empezar a utilizarlo debemos habilitarlo. Para ello, ejecutaremos el comando “systemctl enable” seguido del servicio que acabamos de crear. En nuestro caso, por ejemplo, sería:

```
systemctl enable ejemplo.service
```

Podemos iniciar manualmente el servicio con:

```
systemctl start ejemplo.service
```

O deshabilitar su inicio automático con:

```
systemctl disable ejemplo.service
```

Cada vez que un usuario inicia sesión, se ejecutarán los siguientes ficheros relacionados con el perfil de acceso al sistema:

- /etc/profile: ejecuta el perfil genérico para todos los usuarios.
- /home/nombre_usuario/.profile: ejecuta el .bashrc
- /home/nombre_usuario/.bashrc: contiene comandos que se ejecutan al inicio del Shell de forma interactiva.
- /home/nombre_usuario/.bash_history: almacena el histórico de comandos que introduce el usuario en consola.
- /home/nombre_usuario/.bash_logout: se ejecuta cuando el usuario sale de la sesión.

Hay que destacar que cuando se inicia sesión desde un terminal al cambiar de usuario solamente se ejecuta el fichero .bashrc.

8.5. Políticas de seguridad

Los sistemas GNU/Linux hacen uso de **PAM** (Pluggable Authentication Modules), que es un mecanismo flexible para la autenticación de usuarios centralizado. Mediante estos módulos se pueden modelar políticas de seguridad personalizadas dependiendo del servicio para distintos usuarios.

El directorio **/etc/pam.d/** contiene los archivos de configuración de PAM para cada aplicación tipo PAM. En versiones antiguas de PAM se utilizaba /etc/pam.conf, pero este archivo ya no se utiliza y solamente es usado si el directorio /etc/pam.d/ no existe.

Cada aplicación tipo PAM o servicios tiene un archivo dentro del directorio /etc/pam.d/. Cada uno de estos archivos llevan el nombre del servicio para el cual controla el acceso. Depende del programa tipo PAM definir el nombre de su servicio e instalar su archivo de configuración en el directorio /etc/pam.d/. Por ejemplo, el programa login define su nombre de servicio como login e instala el archivo de configuración PAM /etc/pam.d/login.

Hay cuatro **tipos de módulos** PAM disponibles. Cada uno corresponde con un aspecto diferente del proceso de autorización:

- auth: tareas encaminadas a comprobar el usuario. Es la tarea más genérica y la que más identifica a PAM.
- account: verificar que sea permitido el acceso, por ejemplo permitir o denegar el acceso en función de un horario, recursos o ubicación geográfica.
- password: mantiene actualizado el elemento autenticador de cada usuario, como por ejemplo su contraseña. Se comprueba la fortaleza y que cumpla las políticas definidas, longitud, tipo de caracteres, repeticiones, descartar palabras de diccionario.
- session: configura y administra las sesiones de usuarios. Engloba tareas que se deben llevar a cabo antes del inicio del servicio y cuando finaliza.

Todos los módulos PAM generan un resultado de éxito o fracaso cuando se les llama. Los **indicadores de control** le dicen a PAM qué hacer con el resultado. Como los módulos pueden apilarse en un determinado orden, los indicadores de control le dan la posibilidad de fijar la importancia de un módulo con respecto al objetivo final del proceso de autenticación para el servicio. Hay cuatro indicadores de control definidos:

- required: Indica que es necesario que el módulo tenga éxito para que la pila lo tenga. Si hay fallo no lo veremos hasta que se procese el resto.
- requisite: Es igual que el anterior, pero en caso de fallo el usuario es notificado inmediatamente con un mensaje.
- sufficient: Si en los anteriores no se ha producido fallo, da el visto bueno. A partir de este punto ignora los siguientes required y requisite posibles.
- optional: PAM ignora los módulos marcados por este indicador, salvo si no se llega a ningún valor en concreto de éxito o fracaso.

Por ejemplo, en los sistemas GNU/Linux existe la posibilidad de **limitar recursos** a los usuarios o grupos, como el máximo numero de logins que puede realizar simultáneamente un usuario, el máximo tiempo de CPU, el máximo número de procesos, etc.. Estos límites se controlan a través del fichero **/etc/security/limits.conf**, que es

cargado en el módulo pam_limits.so dentro el archivo de configuración PAM /etc/pam.d/login.

```
# Sets up user limits according to /etc/security/limits.conf
# (Replaces the use of /etc/limits in old login)
session    required   pam_limits.so
```

También es posible **limitar los tiempos de uso de servicios** a los usuarios, para ello podemos hacer uso del módulo **pam_time.so** que podemos cargar en **/etc/pam.d/common-account** añadiendo la siguiente línea:

account requisite pam_time.so

El fichero de configuración del módulo pam_time.so se encuentra en **/etc/security/time.conf.** En este fichero de configuración podemos indicarle al sistema cuándo pueden usar los usuarios los servicios del sistema, cada linea de configuración tiene el siguiente formato:

servicio; terminales; usuarios ; tiempo de acceso

En el fichero de configuración **/etc/security/time.conf** se pueden utilizar caracteres especiales como & (y lógica), | (o lógica), * (comodín), ! (negación). El tiempo de acceso se expresa con dos caracteres para el día de la semana Mo Tu We Th Fr Sa Su Wk Wd Al y cuatro dígitos para la hora, se puede obtener más información consultando la ayuda con ***man time.conf.*** Por ejemplo, si quisiéramos que el usuario “raul” no iniciara sesión en el sistema por las mañanas podríamos denegarle el uso del gestor de sesiones gdm.

gdm-passwd;;raul;!0900-1000*

Dentro del directorio /etc/security existen otros ficheros que permiten establecer ciertos aspectos relativos a la seguridad del sistema:

- access.conf: permite establecer los usuarios que pueden conectarse desde qué equipos y terminales.
- pwquality.conf: establece los requisitos que deben cumplir las contraseñas.
- pam_env.conf: define un entorno estandarizado para usuarios, el cual se establecerá cada vez que se llame al módulo pam_env.
- group.conf: limita la pertenencia a grupos por medio del módulo pam_group

8.6. Cuotas de disco

El **sistema de cuotas** provee un mecanismo de control y uso del espacio de disco duro disponible en un sistema. Se pueden establecer limites en la cantidad de espacio y el número de ficheros de que puede disponer un usuario o grupo.

Para implementar el sistema de cuotas es necesario instalar algún paquete de control de dicho sistema. En Ubuntu hay un paquete denominado **quota** que instala todo lo necesario para implementar todo el sistema.

sudo apt install quota

Una vez instalado tenemos que realizar una serie de pasos para activar el mecanismo de cuotas. Estos pasos son:

1. **Elección del sistema de ficheros** sobre el que se aplican las cuotas: hay que seleccionar que sistema de ficheros necesitan tener aplicadas las cuotas. Lo normal es que solo el sistema donde están las cuentas de usuarios tengan cuotas, aunque es recomendable que tenga cuotas todo sistema de ficheros donde los usuarios puedan escribir. Para habilitar las cuotas en un sistema de ficheros hay que **editar** el fichero **/etc/fstab** e incluir las opciones **usrquota** y **grpquota**:

   */dev/sda5 /home ext4 defaults,**usrquota**,**grpquota** 0 2*

2. **Habilitar las cuotas:** una vez reiniciado el sistema, o remontado el sistema de ficheros , chequeamos y activamos las cuotas con:

sudo quotacheck -ugm /home

u - user, verifica por soporte de cuotas para usuarios.

g - group, verifica por soporte de cuotas para grupos.

m - no-remount, evita que el sistema se remonte como de solo lectura.

sudo quotaon -ug /home

Se puede comprobar que todo funciona bien con:

quotaon -pa

3. **Especificar cuotas** para usuarios o grupos: para editar la cuota de un usuario o grupo se usa el comando **edquota** con la opción **-u** para editar las cuotas de usuarios y con la opción **-g** para editar las opciones de grupo. Sólo hay que editar los números que están detrás de soft y hard.

   sudo edquota -u raul

   - **Filesystem:** Partición restringida.
   - **blocks:** Número de bloques (por un bloque se entiende 1KB de datos) actualmente usados por el usuario.
   - **soft:** Restricción suave de bloques, es decir valor máximo hasta que se dejará rebasar por un tiempo hasta que pasado un tiempo se limite al valor duro. Si vale 0 este límite no se aplicará.
   - **hard:** Restricción dura de bloques a la que limitaremos al usuario, pero que dejaremos rebasar temporalmente hasta el límite suave. Si vale 0 este límite no se aplicará.
   - **inodes:** Número de inodos actualmente en uso. (por un inodo se entiende 1 fichero)
   - **soft:** Restricción suave de inodos, es decir valor máximo hasta que se dejará rebasar por un tiempo hasta que pasado un tiempo se limite al valor duro. Si vale 0 este límite no se aplicará.
   - **hard:** Restricción dura de inodos a la que limitaremos al usuario, pero que dejaremos rebasar temporalmente hasta el límite suave. Si vale 0 este límite no se aplicará.

4. El **período de gracia** que hay entre el límite soft y el hard puede cambiarse con:

   sudo edquota -t

Para verificar las cuotas que tiene un usuario se utiliza el comando:

quota -v

El superusuario puede ver las cuotas de todos los usuarios con el comando:

sudo repquota /home

Para **deshabilitar las cuotas** de un usuario o grupo solo hay que editar las cuotas y poner los limites a 0. Así un usuario puede usar tantos bloques e i-nodos como quiera. Cuando por alguna razón sea necesario también es posible desactivar las cuotas totalmente con el comando **quotaoff.**

quotaoff -v /home

8.7. Resumen de comandos

useradd: permite añadir nuevos usuarios al sistema, además de establecer la información por defecto de los nuevos usuarios que se añadan.

Sintaxis: useradd [opciones] [login]

adduser: en los sistemas Linux actuales se ha creado un script denominado adduser que nos permite crear usuarios de una forma más amistosa.

Sintaxis: adduser [opciones] [login]

groupadd: permite gestionar los grupos del sistema.

Sintaxis: groupadd [opciones] grupo

addgroup: script que permite crear grupos de manera más amistosa.

Sintaxis: addgroup grupo

userdel: permite eliminar definitivamente un usuario del sistema.

Sintaxis: userdel [opciones] <login>

groupdel: permite eliminar definitivamente un grupo del sistema.

Sintaxis: groupdel [opciones] grupo

passwd: permite cambiar el password de un usuario. También puede bloquear, desbloquear y deshabilitar una cuenta. Si se invoca sin argumentos se asume el usuario actual.

Sintaxis: passwd [opciones] [login]

usermod: se emplea para modificar algunas propiedades de los usuarios como: el login, el directorio base, el shell que se inicia al conectarse, los grupos a los que pertenece, la fecha de expiración de la cuenta, etc. También bloquea y desbloquea una cuenta.

Sintaxis: usermod [opciones] <login>

chfn: permite cambiar la información de contacto de un usuario. Esta incluye aspectos como: el nombre completo, la oficina de trabajo y los teléfonos. Se almacena en el fichero de usuarios /etc/passwd.

Sintaxis: chfn [opciones] [login]

su: permite ejecutar un shell (u otro comando) cambiando los identificadores del grupo y del usuario actual. Si no se especifica el login del usuario se asume root.

Sintaxis: su [opciones] [login]

sudo: en algunos sistemas, como puede ser Ubuntu, existe una orden denominada sudo que permite ejecutar una orden como el usuario root.

Sintaxis: sudo [comando]

id: imprime dado un usuario, sus identificadores de usuario y de grupo principal (gid y uid) así como del resto de los grupos a los que pertenece. Sin argumentos se asume el usuario actual.

Sintaxis: id [opciones] [login]

chage: permite gestionar la información sobre la caducidad de las contraseñas.

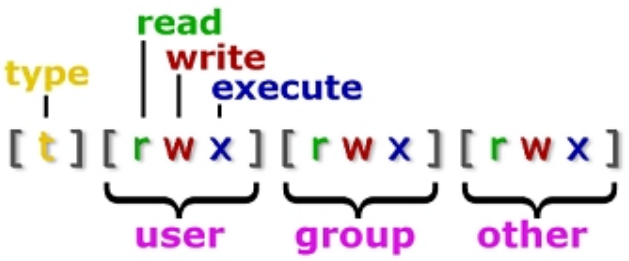
last: indica las últimas conexiones de usuario que han existido en el sistema.

lastb: indica los últimos intentos de conexión fallidos en el sistema.

9. RECURSOS LOCALES. GESTIÓN DE PERMISOS

9.1. Permisos clásicos

Para brindar algo de privacidad y protección cada archivo o directorio tiene asociados **permisos** diferentes para el **propietario**, para el **grupo** y para **los demás usuarios**. En el caso de archivos los permisos que pueden darse o quitarse son: (r) lectura, (w) escritura y (x) ejecución. En el caso de directorios los permisos son: (r) para listar los archivos, (w) para escribir, crear o borrar archivos y (x) para atravesar directorios.



Desde un administrador de archivos, puede ver los permisos de un archivo con el botón derecho del mouse cuando el puntero está sobre el archivo, escogiendo la opción apropiada del menú que aparece. Desde un intérprete de comandos puede emplear el comando ls con la opción -l. Un ejemplo del resultado de este comando se presenta a continuación:

-rwxrwxrwx 1usuario usuario 0 Nov 19 20:13 raul.sh

Los permisos de un archivo pueden ser modificados por el propietario o por el administrador del sistema con el comando **chmod** que espera dos parámetros: cambio por realizar al permiso y nombre del archivo por cambiar. Los permisos se pueden especificar **en octal** o con una o más **letras para identificar al usuario** (**u** para el usuario, **g** para el grupo, **o** para los demás usuarios y **a** para todos), un **+**, un **-** o un **=** y después **letras para identificar los permisos** (**r**, **w** o **x**). Por ejemplo:

chmod og+x raul.sh

Da a los demás usuarios y al grupo permiso de ejecución del archivo raul.sh que debe estar en el directorio desde el cual se da el comando. Cuando no se especifican usuarios chmod toma por defecto todos los usuarios.

El **propietario** de un archivo puede ser modificado sólo por el administrador del sistema con el comando **chown**. Un usuario que pertenezca a varios grupos puede cambiar el grupo de uno de sus archivos a alguno de los grupos a los que pertenezca con el comando **chgrp.** El comando chown también permite cambiar el propietario y el grupo de un fichero de una sola vez:

chown raul:profesores exámenes

Chmod también se puede utilizar **con números**, en este caso el Propietario, Grupo y Otros están representados por tres números, cada numero representa en octal los permisos r(1), w(1), x(1). Para obtener el valor de las opciones para determinar el tipo de acceso necesarios para el archivo hay que agregarlo a continuación. Por ejemplo si deseas que un archivo tenga estos permisos -rwxr-xr-- debes hacer lo siguiente:

Propietario: leer, escribir y ejecutar	*Grupo: leer y ejecutar*	*Otros:leer*
rwx → 111 = 7	*r_x → 101 = 5*	*r_ _ = 4*

chmod 754 archivo

Para cambiar todos los permisos de cada archivo o carpeta bajo un directorio específico de una vez, usa **chmod** con **–R,** modo recursivo.

Habitualmente, los permisos de los archivos en Unix se corresponden con un número en octal que varía entre 000 y 777; sin embargo, existen unos permisos especiales que hacen variar ese número entre 0000 y 7777: se trata de los bits de permanencia o **sticky bit** (1000), **SGID** (2000) y **SUID** (4000).

El bit de **SUID** o **setuid** se activa sobre un fichero añadiéndole **4000** a la representación octal de los permisos del archivo y otorgándole además permiso de ejecución al propietario del mismo; al hacer esto, en lugar de la x en la primera terna de los permisos, aparecerá una **s** o una **S** si no hemos otorgado el permiso de ejecución correspondiente (en este caso el bit no tiene efecto):

*-rw**s**rwxrwx 1 root other 0 Feb 9 17:51 /tmp/file1*

El bit **SUID** activado sobre un fichero indica que todo aquél que ejecute el archivo va a tener durante la ejecución los mismos privilegios que quién lo creó; dicho de otra forma, si el administrador crea un fichero y lo "seguida", todo aquel usuario que lo ejecute va a disponer, hasta que el programa finalice, de un nivel de privilegio total en el sistema.

Todo lo que acabamos de comentar con respecto al bit setuid es aplicable al bit **setgid** pero a nivel de grupo del fichero en lugar de propietario, todo usuario que ejecute un programa "setgidado" tendrá los privilegios del grupo al que pertenece el archivo. Para activar el bit de setgid sumaremos **2000** a la representación octal del permiso del fichero y además habremos de darle permiso de ejecución a la terna de grupo; si lo hacemos, la **s** o **S** aparecerá en lugar de la x en esta terna. Si el fichero es un directorio y no un archivo plano, el bit setgid afecta a los ficheros y subdirectorios que se creen en él: estos tendrán como grupo propietario al mismo que el directorio setgidado, siempre que el proceso que los cree pertenezca a dicho grupo.

*-rwxrw**s**rwx 1 root other 0 Feb 9 17:51 /tmp/file2*

Por otra parte, el **sticky bit** o bit de permanencia se activa sumándole **1000** a la representación octal de los permisos de un determinado archivo y otorgándole además permiso de ejecución; si hacemos esto, veremos que en lugar de una x en la terna correspondiente al resto de usuarios aparece una **t** (si no le hemos dado permiso de ejecución al archivo, aparecerá una **T**):

*-rwxrwxrw**t** 1 root other 0 Feb 9 17:51 /tmp/file3*

Si el bit de permanencia de un fichero está activado le estamos indicando al sistema operativo que se trata de un archivo muy utilizado, por lo que es conveniente que permanezca en memoria principal el mayor tiempo posible; esta opción se utilizaba en sistemas antiguos que disponían de muy poca RAM, pero hoy en día prácticamente no se utiliza. Lo que si que sigue vigente es el efecto del sticky bit activado sobre un directorio: en este caso se indica al sistema operativo que, aunque los permisos "normales" digan que cualquier usuario pueda crear y eliminar ficheros (por ejemplo, un 777 octal), **sólo el propietario de cierto archivo y el administrador pueden borrar un archivo guardado en un directorio con estas características**.

Este bit, que sólo tiene efecto cuando es activado por el administrador (aunque cualquier usuario puede hacer que aparezca una t o una T en sus ficheros y directorios), se utiliza principalmente en directorios del sistema de ficheros en los que interesa que todos puedan escribir pero que no todos puedan borrar los datos escritos, como /tmp/ o /var/tmp/: si el equivalente octal de los permisos de estos directorios fuera simplemente 777 en lugar de 1777, cualquier usuario podría borrar los ficheros del resto.

Si en lugar de especificar el valor octal de los permisos queremos utilizar la **forma simbólica** de chmod, utilizaremos **+t** para activar el bit de permanencia, **g+s** para activar el de setgid y **u+s** para hacer lo mismo con el de setuid; si queremos resetearlos, utilizamos un signo "-" en lugar de un "+" en la línea de órdenes.

Se puede utilizar el comando **umask** para asignar permisos predeterminados a los archivos que creemos. Este comando se utiliza con un valor octal de tres dígitos que representan los permisos que se eliminan de los permisos por defecto de los ficheros. Cuando se crea un archivo, tendrá los mismos permisos que el programa que lo creó menos los permisos que prohíba umask.

Los **permisos por defecto** para los **ficheros** es **666** y para los **directorios 777** y se usa de forma predeterminada la mascara 002, es decir, **umask 002** lo que corresponde al permiso por defecto **664** para los **ficheros**, y **775** para los **directorios**. Por ejemplo, si se desea suprimir el acceso a nuestros ficheros al resto de usuarios, haríamos umask 077.

No se puede obligar a los usuarios a tener un valor determinado de umask porque siempre podrán modificarlo y asignarle el valor que ellos quieran. Aunque se puede asignar un valor por defecto en los archivos que definen el entorno del usuario al iniciar la sesión (como .bashrc, .profile,..)

9.2. Listas de control de acceso. ACL's

Las listas ACL de Linux son una ampliación del modelo estándar de 9 bits. Las listas ACL permiten definir los bits rwx de forma independiente para cualquier combinación de usuarios y grupos.

Para identificar a los usuarios o a los grupos se pueden utilizar tanto su nombre como su identificador UID/GID. El número de entradas que puede contener una lista ACL varía de un sistema de ficheros a otro, y va desde 25 entradas en los sistemas XFS hasta casi ilimitadas en los sistemas ReiserFS o JFS. Los sistemas ext pueden trabajar con 32 entradas, que es un límite más que razonable.

Para conocer los permisos asignados a un directorio/archivo, se emplea el comando: **getfacl** (get file access control list).

Sintaxix: getfacl [opciones] fichero

Para asignación de permisos se emplea el comando: **setfacl** (set file access control list), el cual permite:

- Asignar permisos básicos a: Usuario, Grupo, Otros (UGO). Equivalente a lo que hace el comando: chmod.
- Asignar permisos por defecto (para nuevos directorios/archivos).
- Asignar mascaras.
- Asignar permisos adicionales a grupos y usuarios.

Sintaxis: *setfacl [opciones] [entrada_acl] fichero*

Los parámetros básicos son:

- -R: cambia permisos a archivos y directorios de forma descendente a partir de un directorio dado.
- -d: asigna los permisos por defecto.
- -b: borra todos los permisos adicionales, conservando únicamente los clásicos.
- -k: borra los permisos por defecto.
- -m: modifica los permisos agregando/cambiando por los nuevos valores.
- -x: permite eliminar una serie de entradas de la lista acl.

El formato de las entradas acl son:

- user::permisos → permisos (rwx) para el propietario del archivo.
- user:nom_usuario:permisos → permisos para el usuario nom_usuario.
- group::permisos → permisos para el grupo propietario.
- group:nom_grupo:permisos → permisos para el grupo nom_grupo.
- other::permisos → permisos para otros usuarios.
- mask::permisos → permisos máximos otorgados a los usuarios (excepto al propietario) y a los grupos.

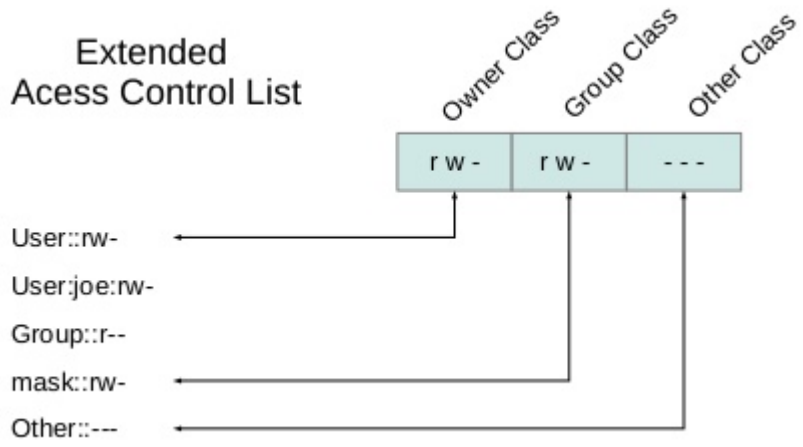
Ejemplo: *setfacl –m user::r, user:raul:rw,group:admin:rwx /tmp/ejemplo*

Si a una entrada acl de un directorio le añado **default** le indico al sistema que asigne esos permisos a todo lo que se cree dentro del directorio, esta entrada no se aplica al directorio actual.

Ejemplo: *setfacl –m **default:**user:raul:rw /tmp/ejemplo2/*

En las entradas acl de Linux **no hay acumulación de permisos**, el permiso más específico es el que se aplica, es decir el **orden de aplicación** sería propietario, usuario, grupo propietario, grupo y otros. Teniendo en cuenta que la máscara (mask) se aplica sobre cualquier entrada, a excepción del propietario y otros. Si a un usuario le especifico unos permisos concretos tendrá esos permisos no los de un grupo al que pertenezca aunque también lo haya añadido en la acl. Si un usuario pertenece a dos grupos secundarios de una acl el sistema tiene en cuenta en primer grupo que aparece en la acl.

Cuando se han establecido acl's, el comando "ls -l" muestra los permisos de propietario, **máscara** y otros, apareciendo a continuación un símbolo "**+**" que indica que hay más permisos establecidos. Hay que prestar atención cuando usamos acl's al comando "chmod XXX" ya que a partir de ahora modifica propietario, **máscara** y otros.



También se pueden instalar herramientas gráficas para la gestión de listas de control de acceso como eiciel.

sudo apt install eiciel

9.3. Recursos compartidos con SAMBA.

Samba es un software que permite compartir archivos e impresoras con otros ordenadores en una misma red local. Utiliza para ello un protocolo conocido como **SMB/ CIFS** compatible con sistemas operativos UNIX o Linux , como Ubuntu, pero además con sistemas Windows (10, 8.1, 7,...), OS/2 o incluso DOS.

Cuando se va a compartir el primer directorio, Ubuntu ofrece instalar el "Servicio de Compartición (Samba)". Pero si se intenta compartir la carpeta se obtendrá un error y no se podrá hacer. La razón es que la instalación de Samba agrega el usuario al grupo sambashare necesario para que se pueda compartir, como Nautilus-Share no se dará cuenta de que el usuario ya está en el grupo sambashare dará error. Para poder empezar a compartir es necesario cerrar y volver a iniciar sesión.

Share permite a través del menú contextual de la carpeta (click derecho) compartirla permitiendo configurar si el acceso es sólo lectura o total y si se permite el acceso a usuarios anónimos. Esa funcionalidad se basa en un método de Samba llamada usershare que permite a los usuarios compartir sus carpetas sin necesitar privilegios de administrador. La configuración para las carpetas compartidas de ésa forma se almacena en el directorio **/var/lib/samba/usershares/** (un archivo por carpeta).

También se puede instalar samba de la forma habitual, usando Añadir y quitar... , Synaptic, o apt:

sudo apt install samba

Samba se compone de varios paquetes entre los cuales destacan los siguientes:

- samba: servidor de archivos e impresoras para Unix.
- samba-common: archivos comunes de samba utilizados para clientes y servidores.
- smbclient: cliente simple para terminal para Unix.
- samba-doc: documentación de Samba.
- smbfs: comandos para montar y desmontar unidades de red samba

Existen **herramientas** para facilitar la configuración del servidor samba como:

- system-config-samba : editor gráfico de configuración.
- gadmin-samba : editor gráfico de configuración Avanzado.
- swat (Samba Web Administration Tool) : permite administrar samba a través del navegador web en http://localhost:901

Aunque lo habitual es configurar el servidor samba editando el archivo **/etc/samba/smb.conf** el cual podemos modificar de la siguiente manera:

sudo gedit /etc/samba/smb.conf

Por ejemplo, para **modificar el grupo de trabajo** buscamos la línea workgroup y modificamos el valor:

workgroup = 1DAM

Para **compartir un directorio** se puede agregar al final del fichero de configuración algo como esto :

[compartido]
comment = Compartido para todos solo con lectura
path = /home/raul/compartido
guest ok= yes
writable = no

Para probar la correcta sintaxis del archivo de configuración se puede utilizar el comando **testparm**, el cual verifica la configuración y despliega las carpetas compartidas en éste:

Puesto que editando el archivo smb.conf se pueden configurar más de 300 parámetros, dando lugar a miles de configuraciones, nos limitaremos a analizar los **parámetros** más relevantes para compartir recursos:

- administrative share : recurso administrativo creado por defecto en sistemas Windows. Por Ejemplo: C$, D$, IPC$.
- admin users : lista de usuarios con total permiso sobre los recursos.
- browsable o browseable : define si el recurso es apreciable en la lista de recursos del servidor .
- comment : permite definir un comentario aplicado al recurso, desplegable en los recursos de toda la red; a través de net view o de smbclient .
- create mode o create mask : permite restringir los permisos de los archivos/directorios creados en un recurso a través de una máscara de permisos .
- force create mode : permite aplicar permisos a los archivos/directorios creados en un recurso .
- directory mode o directory mask : similar a create mode o create mask, aplicado a directorios .
- force directory mode : similar a force create mode, aplicado a directorios .
- directory security mask : mascara de permisos aplicado a clientes Windows.
- group o force group : grupo utilizado como gid del usuario conectado .
- user o force user : usuario utilizado como uid del usuario conectado .
- guest ok : permite acceder a un recurso sin identificarse .
- only guest o guest only : permite acceder sólo a usuarios sin identificación, requiere el parámetro "guest ok" .
- allow hosts o hosts allow : lista de equipos permitidos de acceder al recurso.

- deny hosts o hosts deny : lista de equipos no permitidos de acceder al recurso.
- invalid users : lista de usuarios no permitidos de acceder al recurso .
- only user : permite el acceso sólo a los usuarios en la lista user .
- directory o path : ruta del recurso indicado .
- read list : lista de usuarios que tienen acceso de sólo lectura.
- write list: lista de usuarios que tienen acceso de sólo escritura.
- read only o writeable : no se permite crear ni modificar los archivos/directorios del recurso .
- user , users o username : usuarios con los cuales se probará la contraseña entregada.
- valid users : lista de usuarios permitidos de acceder al recurso .
- invalid users : lista de usuarios no permitidos de acceder al recurso.
- writable o writeable : permite escribir/modificar en el recurso indicado .

Para realizar la **autenticación** de usuarios en el acceso a recursos compartidos mediante samba es necesario que el usuario con el que nos queremos autenticar exista localmente en el sistema y que tenga contraseña en la base de datos de samba.

Se puede hacer lo siguiente para crear un usuario genérico que no tenga acceso local al equipo pero si para acceder a los recursos compartidos estableciendo o no contraseña de acceso:

sudo useradd -s /bin/false usuario

sudo smbpasswd -a usuario

Los usuarios tienen acceso a smbpasswd para cambiar sus contraseñas, también se puede configurar Samba para que sincronice la contraseña, de forma que cuando un usuario cambia la contraseña de cuenta local, la contraseña de Samba es actualizada también:

[global]

unix password sync = yes

Si un usuario no necesita acceso al servidor de Samba durante un largo periodo de tiempo, se puede inhabilitar temporalmente la cuenta y después habilitarla en una fecha posterior. Si un usuario ya no necesita el acceso, se puede suprimir la cuenta:

- Deshabilitar un usuario de samba: *sudo smbpasswd -d raul*
- Habilitar un usuario de samba: *sudo smbpasswd -e raul*
- Eliminar un usuario de samba: sudo smbpasswd -x raul

Una herramienta incluida con la suite de Samba es **pdbedit**, que pertmite gestionar cuentas de samba. Además de crear, modificar y eliminar usuarios de samba, se puede usar pdbedit, entre otras cosas, para:

- Listar cuentas de usuario: *sudo pdbedit -L -v*
- Exportar BD de samba: *pdbedit -e smbpasswd:/home/raul/samba_backup.bak*
- *Importar la BD de samba: pdbedit -i smbpasswd:/home/raul/samba_backup.bak*
- Establecer políticas de cuenta. Los nombres de políticas de cuenta que se puede gestionar son:
  - min password length
  - password history
  - user must logon to change password
  - maximum password age
  - minimum password age
  - lockout duration
  - reset count minutes
  - bad lockout attempt
  - disconnect time
  - refuse machine password change

Con el parámetro -P toma un argumento de cadena de caracteres que debe coincidir exactamente con uno de los nombres de política predefinidos, mientras que el parámetro -C toma un argumento del valor para la configuración de la política, por ejemplo:

sudo pdbedit -P 'min password length' -C 8

sudo pdbedit -P 'maximum password age' -C 30

Un vez establecidos los usuarios de samba podemos compartir recursos con autenticación editando /etc/smb.conf

[accesoseguro]
path = /home/usuario/compartido
valid users = usuario
guest ok = no
writable = yes
browsable = yes

En el apartado “valid users” se puede establecer una lista de usuarios o un grupo anteponiendo el carácter @ por ejemplo:

valid users = @profesores

Después de efectuar algún cambio en el archivo /etc/samba/smb.conf debemos reiniciar samba, para ello :

sudo systemctl restart smbd.service

Los **permisos** de compartición por samba no excluyen a los permisos del sistema Unix sino que **prevalecen los más restrictivos**. Si un directorio está compartida con permiso samba de lectura y escritura pero en los permisos del sistema solo disponemos de permiso de lectura, no podremos escribir. Si una carpeta está compartida con permisos samba de lectura y disponemos de permisos de lectura y escritura en el sistema, tampoco podremos escribir. Para poder escribir necesitaremos disponer permiso de lectura y escritura tanto en los permisos del sistema como en los permisos de compartición samba.

Para **acceder** a los recursos compartidos con samba desde Windows se pueden usar los mismos métodos que para acceder a un recurso compartido de otro Windows. Para acceder a los recursos compartidos desde otro Linux debemos instalar el paquete **smbclient**. Luego, abrimos una carpeta y colocamos en la barra de direcciones:

smb://IP/recurso_compartido/

o desde el shell con el comando **smbclient**:

smbclient //IP/recurso_compartido/ -U usuario

Una vez dentro del smbclient podemos ver los comandos disponibles escribiendo help o ?, dos comandos muy útiles son:

- **put** fichero_local nombre_remoto : para dejar un fichero en el recurso compartido.
- **get** fichero_remoto nombre_local : para copiar en nuestro equipo algún recurso.

9.4. Recursos compartidos con NFS

Cuando tenemos que compartir archivos en una red mixta (Linux y Windows) lo mas indicado es usar Samba. En el caso de una red completamente Linux podemos usar **NFS** (Network file System).

Básicamente NFS permite, a PCs que utilizan Linux, compartir y conectarse a carpetas compartidas entre sí. Es el sistema nativo que utiliza Linux para compartir y acceder a carpetas compartidas en la red. Existen otras alternativas para compartir carpetas en una red como samba, ssh o ftp, pero el sistema recomendado para compartir carpetas entre sistemas Linux es NFS.

Para instalar NFS en Ubuntu usamos el siguiente comando

sudo apt install nfs-common nfs-kernel-server

Una vez que esta instalado editamos el archivo **/etc/exports** para crear las carpetas compartidas. En cada línea del archivo de configuración del servidor NFS /etc/exports, se puede especificar:

- La carpeta que se quiere compartir
- El modo en que se comparte (solo lectura 'ro' o lectura y escritura 'rw')
- Desde qué PC o PCs se permite el acceso (nombre o IP del PC o rango de IPs)

Por ejemplo para compartir la carpeta /home/test tanto para lectura como para escritura y para ser accedida desde cualquier máquina agregamos carpeta

*/home/test *(rw)*

Para compartir la carpeta /home/prueba en modo solo lectura y solo para ser accedida desde la maquina con IP 192.168.1.2 agregamos:

/home/prueba 192.168.1.2(ro)

Los **permisos** de compartición por NFS no excluyen a los permisos del sistema Unix sino que **prevalecen los más restrictivos**. Si una carpeta está compartida con permiso NFS de lectura y escritura pero en los permisos del sistema solo disponemos de permiso de lectura, no podremos escribir. Si una carpeta está compartida con permisos NFS de lectura y disponemos de permisos de lectura y escritura en el sistema, tampoco podremos escribir. Para poder escribir necesitaremos disponer permiso de lectura y escritura tanto en los permisos del sistema como en los permisos de compartición NFS.

Cuando se comparte por NFS, se recomienda restringir al máximo los permisos. Si los usuarios no tienen la necesidad de escribir, debemos compartir con permiso de 'solo lectura'. Si los usuarios solo se conectan desde nuestra red 192.168.1.0/24, debemos permitir el acceso solo desde dicha red.

A continuación se explican algunas opciones generales que se pueden usar con NFS:

- **ro** | **rw**: el directorio será compartido en solo lectura (ro) y es la opción por defecto. Si se especifica (rw) el directorio será compartido en lectura y escritura.
- **sync** | **async**: sync comunica al usuario los cambios realizados sobre los archivos cuando realmente se han ejecutado y es la opción recomendada. La opción async mejora el rendimiento y agiliza el funcionamiento del servicio, pero puede generar archivos corruptos si se produce algún tipo de fallo en el servidor.
- **no_subtree_check**: permite que no se compruebe el camino hasta el directorio que se exporta, útil en el caso de que el usuario no tenga permisos para llegar hasta el directorio exportado.
- **root_squash** | **no_root_squash** | **all_squash:**
  - root_squash indica que un usuario identificado como root tendrá acceso al directorio compartido sólo con privilegios de usuario anónimo. De esta forma se ha degradado al root local con los privilegios mas bajos en el servidor, protegiendo así los archivos en el servidor NFS. Para el resto de usuarios se intenta conservar su UID y GID en el servidor.
  - no_root_squash desactiva la opción anterior, es decir, los accesos realizados como root desde el cliente serán también de root en el servidor NFS.
  - all_squash indica que todos los clientes, incluido root, tendrán acceso al directorio con privilegios de un usuario anónimo. No se mantienen los UID y GID de ningún usuario.

Si se utiliza alguna de las opciones squash podemos indicar cuál el el UID y GID del usuario con el que se quiere que se acceda, en lugar del anónimo. En este caso hemos de indicar a continuación de la opción squash, por ejemplo si queremos que los usuarios clientes se conecten en el servidor como el usuario del servidor con uid y gid 1002 se puede hacer lo siguiente:

(rw,all_squash,anonuid=1002,anongid=1002)

Para que los campos se apliquen hacemos:

sudo service nfs-kernel-server restart

Para chequear las carpetas exportadas podemos usar

sudo exportfs

Para montar la carpeta /home/test compartida en la maquina 192.168.1.1 desde la maquina 192.168.1.2 usamos:

sudo mkdir /mnt/test
sudo mount 192.168.1.1:/home/test /mnt/test

Para desmontar:

sudo umount /mnt/test

Finalmente para montar el ejemplo anterior cada vez que arranquemos la maquina editamos el archivo **/etc/fstab** y agregamos la siguiente linea:

192.168.1.1:/home/test /mnt/test nfs

9.5. Resumen de comandos

chmod: se usa para modificar los permisos de acceso asignados a uno o más archivos. Solo puede cambiar los permisos el propietario del archivo o el administrador del sistema.

Sintaxis: chmod [quien] [op] [permisos]

umask: cuando los archivos o directorios son creados por los usuarios, el sistema asigna por omisión unos permisos de acceso que siempre son los mismos. La asignación automática de permisos, o valores por omisión, puede ser modificada mediante una máscara que elimine permisos.

Sintaxis: umask [nnn]

chown: esta orden sirve para cambiar la propiedad de uno o varios ficheros.

Sintaxis: chown propietario fichero

chgrp: un usuario que pertenezca a varios grupos puede cambiar el grupo de sus archivos a alguno de los grupos a los que pertenezca:

Sintaxis: chgrp grupo fichero

getfacl: permite visualizar la lista de control de acceso o acl de un recurso.

setfacl: permite modificar la lista de control de acceso o acl de un recurso.

testparm: permite comprobar la configuración de samba.

smbclient: permite el acceso a un recurso compartido por samba desde el shell.

exportfs: permite comprobar los recursos compartidos mediante NFS.

10. SHELL SCRIPTS

Los scripts no son más que ficheros de texto ASCII puro, que pueden ser creados con cualquier editor del que dispongamos (vi, nano, gedit, emacs, etc.). Una vez creado el script existen dos formas de ejecutarlo:

- La primera consiste en ejecutarlo con el comando *source fichero*, el cual carga el fichero en la memoria de Bash y lo ejecuta.
- La segunda forma implica poner al fichero el permiso de ejecución y escribir la ruta hasta donde se encuentra el fichero.

Es muy típico en este segundo caso empezar el script poniendo en la primera línea ***#!/bin/bash***, de esta forma indicamos que el script se debe ejecutar con Bash, a pesar de que Bash no fuera el shell por defecto. Otros scripts para otros lenguajes como awk, ksh o perl también se escriben empezando la primera línea por el path del shell a utilizar.

A continuación se enumera el orden de preferencia que sigue Bash a la hora de resolver un símbolo:

1. Alias.
2. Palabras clave (keywords) como function, if o for.
3. Funciones
4. Comandos internos (p.e. cd o type).
5. Scripts y programas ejecutables, para los cuales se sigue el orden en que se dan sus directorios en la variable de entorno PATH.

Aunque no es muy común, a veces conviene cambiar estas órdenes usando los comandos internos ***command***, ***builtin*** y ***enable***. Esto nos permite tener alias y scripts con el mismo nombre y elegir cual de ellos ejecutar.

10.1. Funciones

Las funciones de Bash son una extensión de las funciones que existen desde el Bourne Shell. Éstas, a diferencia de los scripts, se ejecutan dentro de la memoria del propio proceso de Bash, con lo que son más eficientes que ejecutar scripts aparte, pero tienen el inconveniente de que tienen que estar siempre cargadas en la memoria del proceso Bash para poder usarse. Actualmente, debido a la gran cantidad de memoria que tienen los ordenadores, el tener funciones cargadas en la memoria de Bash tiene un coste insignificante.

Para definir una función existen dos formatos:

El estilo del Bourne Shell;

```
function nombrefn
{
•••
comandos bash
•••
}
```

O el estilo del C Shell:

```
nombrefn()
{
•••
comandos bash
•••
}
```

No existe diferencia entre ellos, y usaremos ambos indistintamente. Para borrar una función podemos usar el comando ***unset -f nombrefn***.

Para definir una función podemos escribirla en un fichero y cargarla en el shell usando el comando **source**, o bien definirla directamente en modo interactivo con la orden **function NombreFunción**. Cuando definimos una función se almacena como una variable de entorno (con su valor almacenando la implementación de la función). Para ejecutar la función simplemente escribimos su nombre seguido de argumentos, como cuando ejecutamos un comando. Los argumentos actúan como parámetros de la función.

Podemos ver que funciones tenemos definidas en una sesión usando el comando ***declare -f***. El shell imprime las funciones, y su definición, ordenadas alfabéticamente. Si preferimos obtener sólo el nombre de las funciones podemos usar el comando ***declare -F***.

Si una función tiene el mismo nombre que un script o ejecutable, la función tiene preferencia: Esta regla se ha usado muchas veces para, engañando a los usuarios, violar la integridad del sistema.

10.2. Variables del shell

Principalmente Bash utiliza variables de tipo cadena de caracteres. Esto le diferencia de otros lenguajes de programación donde las variables tienen distintos tipos. Aunque éste es el tipo por defecto de las variables de Bash, más adelante veremos que las variables en Bash también pueden tener otros tipos, como por ejemplo enteros con los que realizar operaciones aritméticas. Las variables se declaran ***NombreVariable=Valor*** y se accede a su valor anteponiendo el carácter **$** al nombre de la variable.

Cuando se asignan valores a las variables podemos utilizar tres tipos de comillas:

- Comillas simples (‘): El intérprete de ordenes interpreta el texto entre comillas simples tal cual, sin sustituir nada.
- Comillas dobles (“): El texto que va entre comillas dobles acepta substituciones de variables, caracteres especiales, etc.
- Comillas invertidas (`): cuando ponemos comillas invertidas estamos especificando que el resultado de la orden se asigne a una variable.

Las variables de los shell scripts son muy simples, ya que no tienen tipo definido ni necesitan ser declaradas antes de poder ser usadas. Para introducir valor en una variable simplemente se usa su nombre, y para obtener el valor de una variable se le antepone un símbolo dólar.

```
#!/bin/bash
SALUDA="Hola Mundo"
echo $SALUDA
```

Pera borrar una variable del intérprete de ordenes utilizamos la orden **unset NombreVariable**.

Para mostrar información de salida tenemos la orden **echo** y para introducir valores a las variables desde la entrada estándar tenemos la orden **read.**

echo: envía un texto a la salida estándar.

Sintaxis: echo [opciones] [Variable o texto]

Algunas opciones son:

- -n: evita que se haga un salto de línea al final del texto.
- -e: permite introducir caracteres especiales como:
  - \a: alerta (campana)
  - \n: nueva línea
  - \t: tabulación
  - \\: barra invertida
  - \nnn: el carácter con el número ASCII nnn en octal.

read: se utiliza para que el usuario pueda dar un valor a una variable de forma interactiva.

Sintaxis: read [opciones] NombreVariable

Algunas opciones son:

- -p "Texto": muestra el "Texto" por la pantalla.

- -n número: lee un número de caracteres sin necesidad de presionar ENTER para finalizar la lectura.
- -s: silencia el echo, no se ve por pantalla lo que el usuario escribe.

Los **parámetros posiciónales** son los encargados de recibir los argumentos de un script y los parámetros de una función. Sus nombres son 1, 2, 3, etc., con lo que para acceder a ellos utilizaremos, el símbolo **$** de la forma $1, $2, $3, etc. Además tenemos el parámetro posicional **$0** que almacena el nombre del script donde se ejecuta. No podemos modificar el valor de las variables posicionales, sólo se pueden leer, si intentamos asignarles un valor se produce un error.

```
#!/bin/bash
# Ejemplo de script que recibe parámetros y los imprime
echo "El script $0"
echo "Recibe los argumentos $1 $2 $3 $4"
```

La variable **$#** almacena el número de argumentos o parámetros recibidos por el script o la función. El valor es de tipo cadena de caracteres, pero más adelante veremos como podemos convertir este valor a número para operar con él. Tanto **$*** como **$@** nos devuelven los argumentos recibidos por el script o función.

Aunque cuando no entrecomillamos $* o $@ no existen diferencias entre usar uno y otro, cuando los encerramos entre comillas débiles existen diferencias que conviene resaltar. La primera es que podemos cambiar el símbolo que usa $* para separar los argumentos indicándolo en la variable de entorno **IFS** (Internal Field Separator), mientras que $@ siempre usa como separador un espacio. Es decir, entrecomillar $* tiene el efecto de que se convierte en una sola palabra, mientras que si entrecomillamos $@ cada argumento es una palabra.

El intérprete de órdenes solo permite acceder a los nueve primeros argumentos de forma directa. Para solucionar este problema tenemos la orden **shift.** Esta sentencia tiene el formato ***shift [n].*** Donde n es el número de desplazamientos a la izquierda que queremos hacer con los argumentos. Si se omite n por defecto vale 1. Así si ejecutamos shift el parámetro $1 coge el valor que tenía $2, el parámetro $2 coge el valor de $3 y así sucesivamente.

Cualquier valor introducido en una variable se considera alfanumérico, así que si realizamos lo siguiente:

```
NUMERO=4
echo NUMERO+3
  #obtendremos por pantalla la cadena de caracteres 4+3.
```

En Linux podemos usar varias expansiones en las líneas de comandos, que son especialmente útiles en los scripts. La primera expansión consiste en usar **$( ).** Esta expansión permite ejecutar lo que se encuentre entre los paréntesis, y devuelve su salida.

```
echo pwd      # escribe por pantalla la palabra pwd
echo $(pwd)   # ejecuta la orden pwd, y escribe por pantalla su resultado.
```

El efecto conseguido con **$(orden)** se puede conseguir también usando la tilde invertida **`orden`.** Otra expansión que podemos usar es **$(( ))** (símbolo dólar pero con dos paréntesis). Los dobles paréntesis podemos sustituirlos si queremos por corchetes **$[ ]**. Esta expansión va a tratar como una expresión aritmética lo que este incluido entre los paréntesis, va a evaluarla y devolvernos su valor.

```
NUMERO=4
echo $(($NUMERO+3))      # sería lo mismo poner echo $[$NUMERO+3]
  #obtenemos en pantalla el valor 7.
```

El comando **let**, que nos permite realizar operaciones aritméticas como la anterior, pero sin tener que usar expansiones ni dolares para las variables.

```
NUMERO=4
let SUMA=NUMERO+3
echo $SUMA
```

Los operadores aritméticos que podemos usar para realizar operaciones son:

- \+ *Suma*
- \- *Resta*

- * *Multiplicación*
- / *División*
- ** *Exponente*
- % *Modulo (resto)*

Por defecto los parámetros posicionales son locales al script o función y no se pueden acceder o modificar desde otra función, son variables **locales**. A diferencia de los parámetros posicionales, el resto de variables que definimos en un script o función son **globales**, es decir una vez definidas en el script son accesibles (y modificables) desde cualquier función.

```
# Ejemplo de variable global
function EstasAqui
{
donde='Dentro de la funcion'
}
donde='En el script'
echo $donde
EstasAqui
echo $donde
```

Al ejecutarlo obtenemos:

```
$ dondeestoy
En el script
Dentro de la funcion
```

Si queremos que una variable no posicional sea local debemos de ponerla el **modificador local**, el cual sólo se puede usar dentro de las funciones (no en los scripts).

```
# Ejemplo de variable global
function EstasAqui
{
local donde='Dentro de la funcion'
}
donde='En el script'
echo $donde
EstasAqui
echo $donde
```

Al ejecutarlo obtenemos:

```
$ dondeestoy En el script En el script
```

10.3. Estructuras condicionales

La sentencia condicional **if** y **else** tiene el siguiente formato:

if [condicion]
then
sentencias
else
sentencias
fi

Este lenguaje nos obliga a que las sentencias estén organizadas con estos retornos de carro, aunque algunas personas prefieren poner los then en la misma línea que los if, para lo cual debemos de usar el separador de comandos, que en Bash es el punto y coma (**;**) así:

If [condicion]; then
sentencias
else
sentencias
fi

La condición es cualquier expresión lógica que produzca un resultado verdadero o falso. Si estamos operando con cadenas alfanuméricas, los operadores que podemos utilizar son los siguientes:

Operadores de comparación de cadenas alfanuméricas	
Cadena1 **=** Cadena2	Verdadero si Cadena1 o Variable1 es IGUAL a Cadena2 o Variable2
Cadena1 **!=** Cadena2	Verdadero si Cadena1 o Variable1 NO es IGUAL a Cadena2 o Variable2
Cadena1 **<** Cadena2	Verdadero si Cadena1 o Variable1 es MENOR a Cadena2 o Variable2
Cadena1 **>** Cadena2	Verdadero si Cadena1 o Variable1 es MAYOR a Cadena2 o Variable2
-n Variable1	Verdadero si Cadena1 o Variable1 NO ES NULO (tiene algún valor)
-z Variable1	Verdadero si Cadena1 o Variable1 ES NULO (esta vacía o no definida)

Los anteriores operadores sólo son validos para comparar cadenas, si queremos comparar valores numéricos, hemos de usar los siguientes:

Operadores de comparación de valores numéricos.	
Numero1 **-eq** Numero2	Verdadero si Numero1 o Variable1 es IGUAL a Numero2 o Variable2
Numero1 **-ne** Numero2	Verdadero si Numero1 o Variable1 NO es IGUAL a Numero2 o Variable2
Numero1 **-lt** Numero2	Verdadero si Numero1 o Variable1 es MENOR a Numero2 o Variable2
Numero1 **-gt** Numero2	Verdadero si Numero1 o Variable1 es MAYOR a Numero2 o Variable2
Numero1 **-le** Numero2	Ver. si Numero1 o Variable1 es MENOR O IGUAL a Numero2 o Variable2
Numero1 **-ge** Numero2	Ver. si Numero1 o Variable1 es MAYOR O IGUAL a Numero2 o Variable2

Veamos un ejemplo de una estructura if, tanto con valores de cadena como con valores numéricos.

```
NOMBRE=”Raúl”
if [ $NOMBRE = “Raúl” ]; then
  echo Bienvenido Raúl
fi


NUMERO=12
if [ $NUMERO -eq 12 ]; then
  echo Efectivamente, el número es 12
fi
```

La estructura if podemos ampliarla usando la construcción else (en caso contrario) y **elif** (en caso contrario si…). Una estructura con elif (else if) tiene la siguiente forma:

```
if [ expresión1 ]; then
  realizar si expresión1 es verdadera
elif [ expresión2 ]; then
  realizar si expresión1 es falsa, pero es verdadera expresión2
elif [ expresión3 ]; then
  realizar si exp1 y exp2 son falsas, pero es verdadera expresión3
else
  realizar si todas las expresiones anteriores son falsas
fi
```

Hay que recordar que **los corchetes llevan espacios en blanco** tanto a izquierda como derecha, que el punto y coma sin embargo va pegado al corchete cerrado, y que SIEMPRE hay que poner espacios en blanco en las condiciones.

Hemos visto operadores aritméticos y operadores para cadena, pero en las expresiones podemos utilizar cualquier operación que nos devuelva un valor lógico (0 para verdadero). Por ejemplo, podemos usar la función **test** del bash, que funciona de la siguiente forma:

Operaciones condicionales usando test.	
-a fichero	Verdadero si fichero existe
-d fichero	Verdadero si fichero existe, y es un fichero de tipo directorio
-f fichero	Verdadero si fichero existe, y es un fichero regular.
-r fichero	Verdadero si fichero existe y se puede leer
-w fichero	Verdadero si fichero existe y se puede escribir
-x fichero	Verdadero si fichero existe y se puede ejecutar
-s fichero	Verdadero si fichero existe y no está vacio
fichero1 -nt fichero2	Verdadero si fichero1 es más nuevo que fichero2
fichero1 -ot fichero2	Verdadero si fichero1 es más viejo que fichero2

Si lo necesitamos, podemos anidar expresiones usando tanto **&&** (AND) como **||** (OR). También podemos usar el operador **!** (NOT) para indicar una negación.

Hemos visto la principal estructura condicional que es el if, pero tenemos alguna otra a nuestra disposición, como el **case**.

```
case VARIABLE in
  valor1)
      Se ejecuta si VARIABLE tiene el valor1
      ;;
  valor2)
      Se ejecuta si VARIABLE tiene el valor2
      ;;
  *)
      Se ejecuta si VARIABLE no tiene ninguno de los valores anteriores
      ;;
esac
```

10.4. Bucles

Las principales estructuras iterativas que podemos usar en shell scripts son **for**, **while**, **until**, y **select**.

La estructura **for** es la siguiente:

```
for variable in conjunto; do
  estas líneas se repiten una vez por cada elemento del conjunto,
  y variable va tomando los valores del conjunto
done
```

Ese conjunto que aparece en la estructura del for, es normalmente un conjunto de valores cualesquiera, separados por espacios en blanco o retornos de linea. Así, si queremos mostrar los días de la semana por pantalla este script lo haría:

```
#!/bin/bash
for dia in lunes martes miércoles jueves viernes sabado domingo; do
  echo el dia de la semana procesado es $dia
done
```

Existe una orden interesante en Linux que es **seq,** que nos permite mostrar una secuencia de números, cuyo formato es ***seq primero incremento último***. Esto nos permite realizar este tipo de estructuras con for.

```
#!/bin/bash
for numero in $( seq 1 1 20 ); do
  echo “Número vale :” $NUMERO
done
```

Cuando no queremos recorrer un conjunto de valores, sino repetir algo mientras se cumpla una condición, o hasta que se cumpla una condición, podemos usar las estructuras **while** y **until**.

La estructura del **while** es la siguiente:

```
while [ expresión ]; do
  estas líneas se repiten MIENTRAS la expresión sea verdadera
done
```

La estructura del **until** es la siguiente:

```
until [ expresión ]; do
  estas líneas se repiten HASTA que la expresión sea verdadera
done
```

Ambas estructuras, tanto while como until realmente realizan exactamente lo mismo, al efectuar la comprobación de la expresión en la primera línea, no como en otros lenguajes. La única diferencia entre while y until es que while se ejecuta mientras que el código de terminación del comando sea exitoso, es decir 0, mientras que until se ejecuta hasta que el código de terminación sea exitoso, según esto until se puede interpretar como ejecutar varias veces un comando hasta que tenga éxito.

El bucle “while” es el método más apropiado y simple para leer un archivo línea por línea.

```
while read linea
do
  comando
done < archivo
```

También podemos a partir de un archivo estructurado obtener los valores de cada campo y asignarlos a varias variables con el comando “**read**”. Sin embargo hay que tener cuidado de asignar a la variable “**IFS**” el separador de campo adecuado (el *espacio es el separador por defecto*).

```
while [IFS=separador] read campo1 campo2
do
  comando
done < archivo
```

La última estructura iterativa que vamos a ver es **select**. Esta nos permite realizar una iteración o bucle, pero presentando un menú por pantalla para que el usuario escoja una opción. Su estructura general es la siguiente:

```
select VARIABLE in conjunto_opciones; do
  Aquí variable toma el valor de una de las opciones del conjunto
done
```

Esta estructura como vemos es muy parecida a la del for, pero presenta la principal diferencia en que por definición se crea un bucle sin final, no hay un valor inicial y un valor limite, el bucle se repetirá eternamente, lo que nos obliga a salir, bien con **break** que nos permite salir del bucle o con **exit** que nos permite salir del script entero.

10.5. Arrays

Un **array** o matriz, es una variable que contiene múltiples valores que pueden ser de un mismo tipo o de diferentes tipos. Los arrays en bash pueden ser **indexados**, donde cada valor está relacionado con un índice que por defecto comienza por cero, o pueden ser **asociativos**, donde cada valor está asociado a una clave.

Para declarar una matriz indexada sin inicializarla se utiliza la notación:

declare -a ARRAY

Para declarar una matriz indexada e inicializarla vacía, o para vaciar una matriz existente, se puede usar la notación:

ARRAY=( )

También se puede establecer el primer elemento de una matriz indexada y si no existía antes, se crea:

ARRAY[0]="valor"

A continuación, se repasaran algunas operaciones básicas con arrays en bash :

- Acceder a un elemento concreto del array: *${ARRAY[índice]}*
- Obtener todos los valores del array: *${ARRAY[@]}*
- *Obtener todos los índices del array: ${!ARRAY[@]}*
- Obtener la longitud del array: *${#ARRAY[@]}*
- Buscar y reemplazar elementos en un array: *${ARRAY[@]/original/reemplazo}*
- *Añadir un elemento especificando el índice: ARRAY[índice]="valor"*
- *Añadir un elemento al array sin especificar el índice: ARRAY+=("valor")*
- *Eliminar un elemento del array: unset ARRAY[índice]*

La única marera de declarar un **array asociativo** es:

```
declare -A ARRAY
```

Las operaciones básicas sobre **arrays asociativos** son similares a las de los arrays indexados, teniendo en cuenta que los elementos se almacenan en el array con el par (clave,valor).

Ejemplo de uso de arrays asociativos:

```
#!/bin/bash
#Array asociativo de pareja (clave, valor).
declare -A EQUIPOS
EQUIPOS[host1]="192.168.1.1"
EQUIPOS[host2]="192.168.1.2"
EQUIPOS[host3]="192.168.1.3"
EQUIPOS[host4]="192.168.1.4"
EQUIPOS[host5]="192.168.1.5"

# Número de equipos
echo "Número de PCs: ${#EQUIPOS[@]}"

# Ver todas las IP's
echo "IPS usadas: ${EQUIPOS[@]}"

# Ver todos los equipos
echo "PCs: ${!EQUIPOS[@]}"

# Añade mi equipo
EQUIPOS[`hostname`]="`hostname -I`"

# Recorrer todos los equipos mostrando su IP
for pc in "${!EQUIPOS[@]}"; do
    echo "El equipo $pc tiene la IP ${EQUIPOS[$pc]}"
done
```

10.6. Interfaz gráfica de usuario. Zenity

Zenity, Zen Dialogs, es un software que permite generar diálogos simples con interfaz gráfica empleando la biblioteca de GTK+. Zenity permite crear los siguientes tipos de diálogos simples usando distintas opciones:

- Calendario: --calendar
- Seleccionador de archivos: --file-selection
- Formularios: --forms
- Lista: --list
- Icono de notificación: --notification
- Mensajes:
  - Error: --error
  - Información: --info
  - Pregunta: --question
  - Advertencia: --warning
- Contraseña: --password
- Progreso: --progress
- Entrada de texto: --entry
- Información de texto: --text-info
- Escala: --scale
- Selección de color: --color-selection

Todos los diálogos de Zenity soportan las siguientes opciones generales:

- --title=título. Especifica el título de un diálogo.
- --window-icon=ruta_al_icono. Especifica el icono que se muestra en el marco de la ventana del diálogo. Hay 4 iconos disponibles, proporcionando las palabras claves siguientes: 'info', 'warning', 'question' y 'error'.
- --width=anchura. Especifica el ancho del diálogo.

- --height=altura. Especifica la altura del diálogo.
- --timeout=tiempo_de_expiración. Especifica el tiempo de expiración en segundos después del cual el diálogo se cierra.

Zenity proporciona las siguientes opciones de ayuda:

- --help. Muestra un texto de ayuda abreviado.
- --help-all. Muestra un texto de ayuda completo para todos los diálogos.
- --help-general. Muestra el texto de ayuda para los opciones de diálogo generales.
- --help-calendar. Muestra el texto de ayuda para las opciones de diálogo del calendario.
- --help-entry. Muestra el texto de ayuda para las opciones del diálogo de entrada de texto.
- --help-error. Muestra el texto de ayuda para las opciones del diálogo de error.
- --help-info. Muestra el texto de ayuda para las opciones del diálogo de información.
- --help-file-selection. Muestra el texto de ayuda para las opciones del diálogo de selección de archivos.
- --help-list. Muestra el texto de ayuda para las opciones del diálogo de lista.
- --help-notification. Muestra el texto de ayuda para las opciones de iconos de notificación.
- --help-progress. Muestra el texto de ayuda para las opciones del diálogo de progreso.
- --help-question. Muestra el texto de ayuda para las opciones del diálogo de pregunta.
- --help-warning. Muestra el texto de ayuda para las opciones del diálogo de advertencia.
- --help-text-info. Muestra el texto de ayuda para las opciones del diálogo de información.
- --help-misc. Muestra el texto de ayuda para las opciones misceláneas.

- --help-gtk. Muestra la ayuda para las opciones de GTK+.

Zenity devuelve los siguientes códigos de salida, que se pueden capturar consultando $?:

- 0. El usuario ha presionado Aceptar o Cerrar.
- 1. El usuario ha presionado Cancelar, o ha utilizado la función de la ventana para cerrar el diálogo.
- -1. Ha ocurrido un error inesperado.
- 5. El diálogo se ha cerrado porque se alcanzó el tiempo de expiración.

Ejemplo de uso de Zenity:

```
#!/bin/bash
# Ejempo de script con zenity

nombre=`zenity --entry --title="Saludar" --text="¿Cómo te llamas?"`

if [[ $? -eq 1 ]]; then
   #  cancel
   zenity --warning --text="MALEDUCADO!!!!"
   zenity --progress --title="Formatenado..." --text="Formatenado el disco duro..." --pulsate
else
   zenity --info --title="SALUDA" --text="HOLA $nombre, yo me llamo `hostname`."
fi
```