# TEMA 10: ADMINISTRACIÓN DE LINUX

## Índice

# 1. Introducción

En el tema anterior vimos cómo el sistema operativo Linux se encarga de gestionar la información. Estudiamos los comandos básicos para movernos e interactuar con el sistema de ficheros.

Una vez conocidos esos comandos básicos, en este tema nos vamos a centrar en lo que es la administración del sistema relativa a los procesos, a los usuarios y a la gestión de seguridad de los archivos. Veremos también cómo planificar tareas, instalar y actualizar software y gestionar los procesos en segundo plano que ejecuta el sistema operativo llamados demonios.

El sistema operativo Linux está pensado para poder administrarlo desde la línea de texto, siendo la interfaz gráfica una capa añadida que facilita ciertas tareas. Nosotros nos vamos a centrar en la administración desde la línea de comandos, asumiendo que hacerlo mediante la GUI es más fácil y trivial si sabemos hacerlo desde el intérprete de comandos.

A diferencia de Windows que contaba para almacenar la configuración con el registro, en Linux, toda la configuración del sistema se almacena en ficheros de texto editables por el usuario. La ventaja del sistema Windows es que toda la información estaba centralizada en una base de datos y en cambio en Linux está diseminada en ficheros de texto por todo el sistema de ficheros. La ventaja del sistema Linux es que sabiendo cuál es el fichero a tocar, es fácilmente modificable.

# 2. Gestión de usuarios

Como ya sabemos, Linux es un sistema multiusuario, lo cual permite que varios usuarios puedan conectarse y trabajar en él de forma simultánea.

La gestión de usuarios es una parte crítica en el mantenimiento de la seguridad de un sistema. La gestión ineficaz de usuarios y privilegios casi siempre termina comprometiendo a muchos sistemas. Por esta razón, es importante entender cómo se

puede proteger el servidor por medio de técnicas sencillas y efectivas para la gestión de cuentas de usuario.

Las cuentas de usuario de Linux existen para los usuarios que tienen que acceder al sistema, pero también pueden ser para servicios del sistema (demonios, que veremos a continuación). Aunque un demonio no inicie sesión en Linux, necesita una cuenta. También puede haber otro tipo de cuentas especiales creadas para funciones concretas.

Un usuario se caracteriza por su **login** el cual debe indicar para conectarse al sistema, además de su **password** o contraseña. Además puede poseer un conjunto de datos adicionales mencionados más adelante.

El usuario con más privilegios en Linux es aquel cuyo *login* es **root**. Este es el único con derechos suficientes para acceder a todo el sistema de ficheros sin ninguna restricción.

Los desarrolladores de Ubuntu tomaron una decisión concienzuda para desactivar por defecto la cuenta administrativa *root* en todas las instalaciones de Ubuntu. Esto no significa que la cuenta *root* haya sido eliminada o que no se pueda acceder a ella. Simplemente se le ha dado una contraseña que no concuerda con ningún valor cifrado posible, por esta razón no puede identificarse por sí misma directamente.

En su lugar, se anima a los usuarios a hacer uso de una herramienta llamada **sudo** para realizar tareas administrativas en el sistema. *Sudo* le permite a un usuario autorizado elevar sus privilegios temporalmente usando su propia contraseña en lugar de tener que saber la contraseña perteneciente a la cuenta *root*. Esta metodología sencilla pero efectiva le proporciona responsabilidad a todas las acciones del usuario, y le da al administrador un control puntual sobre las acciones que puede realizar un usuario con dichos privilegios.

Si deseamos activar la cuenta de *root* bastaría con asignarle una contraseña válida a este usuario mediante el comando *passwd* que veremos más adelante.

En Linux además existen grupos de usuarios también administrados por *root* o por un usuario designado por este. Los grupos permiten otorgar los mismos privilegios a un conjunto de usuarios.

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

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, podemos ejecutar la orden **id**.

```
$ id
uid=502(pepe) gid=502(pepe) groups=502(pepe),100(users)
```

Al añadir un usuario también se creará un directorio base para el mismo con el nombre de su *login*. Este directorio se coloca por defecto en el directorio */home* excepto para *root*, cuyo directorio base es */root*.

La información asociada a los usuarios en un sistema Linux se guarda en el fichero **/etc/passwd** y las contraseñas y datos afines en **/etc/shadow**. Por su parte la información de los grupos, sus miembros y passwords están en **/etc/group** y **/etc/gshadow** respectivamente.

El fichero */etc/passwd* puede ser visualizado pero sólo *root* puede modificarlo (o un usuario administrador elevando sus privilegios). Una salida típica de este fichero sería:

```
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:
vicente:x:100:100:Vicente Valverde,,,:/home/vicente:/bin/bash
antonio:x:101:101:Antonio Antúnez,,,:/home/antonio:/bin/bash
alumno:x:102:102:Alumno de ASIR,,,:/home/alumno:/bin/bash
```

En este fichero cada línea representa a un usuario. La información de cada usuario está dividida en 6 campos delimitados por el carácter dos puntos (:). La descripción de cada uno de los campos es la siguiente:

- **Nombre del usuario**. Identificador de inicio de sesión.
- **Contraseña**. Si hay una x significa que se encuentra almacenada en el fichero /etc/shadow
- **UID**. Identificador de usuario.
- **GID**. Identificador del grupo principal al que pertenece el usuario. Los grupos se almacenan en /etc/group
- **Comentarios**. Normalmente la descripción del usuario (nombre comleto, teléfono, etc)
- **Directorio $HOME del usuario**.
- **Shell** que utilizará el usuario.

El fichero donde se almacenan las contraseñas es el fichero **/etc/shadow**. Además de las contraseñas cifradas también se almacena otra información de seguridad (caducidad de la cuenta, cambios de contraseña, etc)

Un posible contenido de este fichero sería:

```
root:Rb3hHmtAg66sPjawYH72MV:10568:0:99999:7:7:99999:1073897392
bin:*:10542:0::7:7::
vicente:hgjYTjhgD6S2jawYH72MV:10573:0:99999:7:7:99999:1073897392
antonio:KJlgkslkj6Akjh44hhkju:10576:0:99999:7:7:99999:1073897392
alumno:fKUKfSeuokf5Dj73HJJk8:10605:0:99999:7:7:99999:1073897392
```

Al igual que */etc/passwd*, el fichero */etc/shadow* está formado por campos de una línea delimitados por el símbolo dos puntos.

Algunos campos parecen números extraños. Para ser interpretados es necesario saber que están referidos a los días transcurridos desde el comienzo de la 'era de la computación' es decir el 01/01/1970. El significado de cada campo es el siguiente:

- Nombre de la cuenta de usuario.
- Contraseña cifrada.
- Días transcurridos desde el comienzo de la era hasta que la contraseña fue cambiada por última vez.

- Número de días que deben pasar hasta que se vuelva a cambiar la contraseña.
- Número de días tras los que hay que volver a cambiar la contraseña (porque caduca).
- Días antes que hay que avisar al usuario antes de que caduque la contraseña.
- Días después de la caducidad de la contraseña que se inhabilitará la cuenta si no se ha cambiado.
- Días transcurridos desde la era en que caducará la cuenta y se inhabilitará.
- Campo reservado.

## *Comandos relacionados con la gestión de usuarios*

### Comando useradd

El comando *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]

Algunas opciones:
-g: Grupo principal que queremos tenga el usuario (debe existir)
-d: Carpeta home del usuario. Suele ser */home/nombre-usuario*
-m: Crear carpeta $HOME si es que no existe.
-s: Intérprete de comandos (shell) del usuario. Suele ser */bin/bash*

La carpeta $HOME del usuario se creará según lo que haya contenido en */etc/skel*. Si no se especifica el parámetro -m el comportamiento por defecto es que no se cree la carpeta personal. Podemos alterar este comportamiento por defecto accediendo al fichero */etc/login.defs* y añadiendo o modificando una línea con:

```
CREATE_HOME yes
```

Ejemplos:

```
# useradd pepe          #crea el usuario pepe con todas las
                        opciones por defecto.
# useradd -D            #muestra las opciones por defecto que se
                        aplicarán a los usuarios nuevos.
# useradd -g profesores -d /home/pedro -m -s /bin/bash pedro
```

## Comando adduser

La orden *useradd* no se suele usar, ya que no configura correctamente las cuentas de usuario. En su lugar en los sistemas Linux actuales se ha creado un script denominado *adduser* que nos permite crear usuarios de una forma más amistosa, pero no en todas las distribuciones funciona.

Sintaxis: adduser [opciones] [login]

adduser usuario grupo

Ejemplos:

```
# adduser pepe          #crea un usuario pepe, preguntándonos por su
                        contraseña, comentarios, etc.
# adduser               #crea un usuario, preguntándonos por su nombre,
                        su contraseña, comentarios, etc.
# adduser pepe cdrom    #añade al usuario pepe, que debe existir ya, al
                        grupo cdrom.
```

## Comando userdel

El comando *userdel* permite eliminar definitivamente un usuario del sistema.

Sintaxis: userdel [opciones] <login>

Algunas opciones:

-r: Elimina todo el directorio base ($HOME del usuario) al eliminar el usuario

Ejemplo:

```
# userdel -r pepe       #elimina al usuario pepe y borra su directorio
                        base. Por defecto el directorio base se
                        mantiene
```

## Comando passwd

El comando *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]

Algunas opciones:

-d: deshabilita la cuenta

-l: bloquea (lock) la cuenta

-u: desbloquea (unlock) la cuenta

Ejemplos:

```
# passwd pepe           #coloca una contraseña para pepe
# passwd -d pepe        #deshabilita la cuenta del usuario pepe
                        eliminando su password
# passwd -l pepe        #bloquea la cuenta del usuario pepe poniendo un
                        signo ! delante de su password en el fichero
                        /etc/shadow
# passwd -u pepe        #desbloquea la cuenta del usuario pepe
$ sudo passwd root      #Asigna una nueva contraseña al usuario root
                        Activándolo si no lo estaba
```

## Comando usermod

El comando *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>

Algunas opciones:

-s: Establece el shell para un usuario

-G: Añade al usuario a los grupos indicados como grupos secundarios (sólo pertenecerá a los grupos indicados)

-a: Se utiliza junto a –G para añadir los grupos indicados como secundarios a los ya existentes. IMPORTANTE

-e: indica la fecha de la expiración de la cuenta

Ejemplos:

```
# usermod -s /bin/csh pepe          #coloca el shell csh para el
                                    usuario pepe
```

```
# usermod -G users,disk pepe        #señala como grupos secundarios de
                                    pepe a users y disk
# usermod -a -G users,disk pepe     #señala como grupos secundarios de
                                    pepe a users y disk además de los
                                    que ya perteneciera
# usermod -e 2018-10-20 pepe        #indica que la cuenta de pepe
                                    expirará el 20 de octubre del 2018
```

## Comando chfn

El comando *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]

Ejemplo:

```
# chfn pepe
```

## Comando su

El comando *su* permite ejecutar un shell (u otro comando) cambiando los identificadores del grupo y del usuario actual. Si se le pasa - como primer argumento ejecuta el shell como un login shell, es decir, se creará un proceso de login tal y como ocurre naturalmente cuando un usuario se conecta al sistema. Si no se especifica el login del usuario se asume *root*.

Para volver a nuestro usuario normal, teclearemos el comando **exit**

Sintaxis: su [opciones] [login]

Ejemplos:

```
$ su        # cambiamos el usuario a root. Pedirá contraseña de root
$ su -      # el guion permite abrir una shell para el usuario.
# su pepe   # cambiamos el usuario a pepe
```

## Comando sudo

En algunos sistemas, como puede ser Ubuntu, existe una orden denominada *sudo*, que viene a ser un `su -c` (es decir, permite ejecutar una orden como el usuario *root*).

Sintaxis: sudo comando

Ejemplos:

```
$ sudo gedit /etc/passwd      # ejecuta gedit /etc/passwd como si lo
                              ejecutara el usuario root
$ sudo cat /etc/shadow        # ejecuta cat /etc/shadow como si lo
                              ejecutara el usuario root
$ sudo adduser                # ejecuta el comando adduser como root
$ sudo !!                     # ejecuta el comando tecleado
                              inmediatamente anterior pero anteponiendo
                              sudo (por si habíamos olvidado ponerlo)
```

Vemos que con el mandato *sudo* podemos elevar nuestros privilegios al de *root* y ejecutar cualquier comando simplemente poniendo nuestra contraseña.

Entonces, ¿eso significa que cualquier usuario puede ejecutar todos los comandos con privilegios de administrador simplemente poniendo *sudo* y su contraseña?

**No**. Sólo pueden hacerlo los administradores. Aunque esto no es exacto. Realmente pueden hacerlo los usuarios o grupos que se le indiquen en el fichero **/etc/sudoers**

Si visualizamos este fichero:

```
$ sudo cat /etc/sudoers
#
# This file MUST be edited with the 'visudo' command as root.
#
# Please consider adding local content in /etc/sudoers.d/ instead of
# directly modifying this file.
#
# See the man page for details on how to write a sudoers file.
#
Defaults    env_reset
Defaults    mail_badpass
Defaults
      secure_path="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/
sbin:/bin"
# Host alias specification
# User alias specification
# Cmnd alias specification
# User privilege specification
```

```
root  ALL=(ALL:ALL) ALL

# Members of the admin group may gain root privileges
%admin ALL=(ALL) ALL

# Allow members of group sudo to execute any command
%sudo ALL=(ALL:ALL) ALL

# See sudoers(5) for more information on "#include" directives:

#includedir /etc/sudoers.d
```

En este fichero se le está indicando que los usuarios miembros del grupo *sudo* y del grupo *admin* pueden adquirir privilegios de *root* y ejecutar cualquier comando.

## Comando id

El 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]

Ejemplo:

```
# id pepe
uid=502(pepe) gid=502(pepe) groups=502(pepe),100(users)
```

## Comando chage

El comando *chage* nos permita gestionar la información sobre la caducidad de las contraseñas.

```
# chage pepe
```

## Comando last

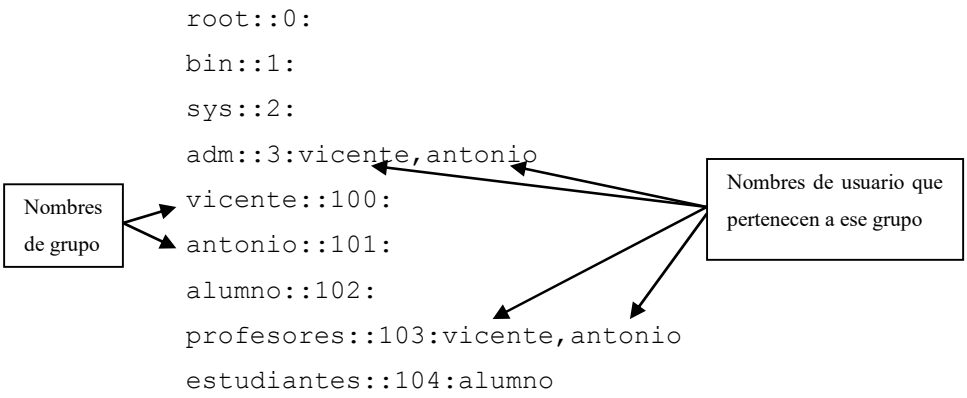
El comando *last* nos indica las últimas conexiones de usuario que han existido en el sistema.

```
# last
```

# 3. Gestión de grupos

Al igual que los usuarios, los grupos tienen un fichero donde se almacena sus datos (nombre de grupo y GID) y los miembros del grupo. El fichero en el que se almacenan estos datos referentes a grupos es el **/etc/group**

Un posible contenido de este fichero sería:




Al igual que en los ficheros anteriores, cada campo, delimitado por el carácter dos puntos (:) tiene un significado:

- Nombre del grupo
- Contraseña del grupo.
- GID. Identificador del grupo
- Cuentas de usuario asociadas a ese grupo (y que no aparecen como su grupo principal)

**Comando addgroup**

Nos permite crear grupos.

Sintaxis: addgroup grupo

Ejemplo:

```
# addgroup alumnos      #Se crea un nuevo grupo llamado alumnos
```

Para añadir un usuario a un grupo utilizamos alguno de los mandatos ya vistos como *adduser* o *usermod* o el aún no visto *gpasswd*

```
# adduser pepe alumnos  #añade al usuario pepe, que debe existir ya, al
                        grupo alumnos que hemos creado.
```

```
# usermod -a -G alumnos pedro      #añade al usuario pedro, que debe
                                   existir ya, al grupo alumnos que
                                   hemos creado.
```

**Comando groupdel**

El comando *groupdel* permite eliminar definitivamente un grupo del sistema.

Sintaxis: groupdel [opciones] grupo

Ejemplo:

```
# groupdel alumnos      # Elimina el grupo alumnos
```

**Comando gpasswd**

El comando *gpasswd* permite administrar los grupos. Se puede utilizar también para añadir y eliminar usuarios (como los vistos anteriormente), señalar un administrador e indicar un password de grupo. Este password de grupo ya no se utiliza. Su objetivo era permitir al usuario cambiar temporalmente su grupo principal por otro al que no pertenecía mediante el comando *newgrp*.

Sintaxis: gpasswd [opciones] <grupo>

Ejemplos:

```
# gpasswd -A pepe admin       # señala como administrador del grupo
                              admin al usuario pepe
$ gpasswd admin               # cambia el passwd del grupo admin
$ gpasswd -a jose admin       # añade el usuario jose al grupo admin
```

**Comando groups**

El comando *groups* permite mostrar todos los grupos a los que pertenece un usuario (primario y secundarios)..

Sintaxis: groups [opciones] [usuario]

Ejemplo:

```
$ groups alumno          # Visualiza los grupos del usuario alumno
alumno adm cdrom sudo plugdev lpadmin sambashare alumnos estudiantes
```

# 4. Atributos y permisos de archivos y directorios

La gestión de permisos en Linux (proceso de autorización para uso de recursos) es bastante distinta de la que hemos estudiado anteriormente en Windows. Linux usa un esquema de permisos bastante más simple, ya que las ACL (listas de control de acceso) de los recursos sólo admiten 3 tipos de usuarios, y básicamente 3 tipos de permisos. Sin embargo, aunque es un sistema muy simple, con una buena planificación permite desarrollar políticas de seguridad lo suficientemente buenas para el uso cotidiano de un sistema.

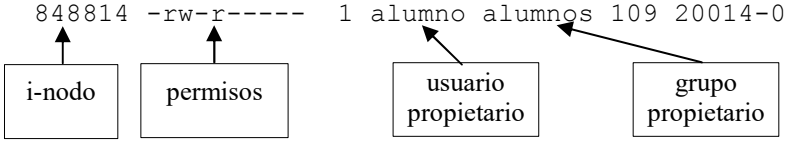
En Linux cada recurso pertenece a un usuario, y a un grupo de usuarios. Podemos saber quién es el usuario y grupo propietario, así como sus permisos mediante el comando `ls` con la opción `-l` (formato largo).

Si ejecutamos el siguiente mandato:

```
$ ls -lia
```

Obtenemos una salida por pantalla de los ficheros de un directorio con el siguiente formato:

```
848814 -rw-r-----  1 alumno alumnos 109 20014-02-01 20:05 fichero
```




Podemos cambiar estas pertenencias con la orden **chown** y **chgrp** (ver más adelante).

**chown usuario:grupo recurso**

**chgrp grupo recurso**

Ejemplo:

```
# chown pepe.profesores fichero
# chgrp profesores fichero
```

Respecto a la columna de permisos que nos muestra el comando `ls -l`, ésta consta de 10 caracteres y se divide de la siguiente manera:

- 1º carácter: Nos indica que tipo de fichero es, los valores posibles para este carácter son:

- d : Directorio.
- l : Enlace simbólico.
- b : Archivo que corresponde a un dispositivo HW de bloque.
- c : Archivo que corresponde a un dispositivo HW de caracteres .
- p : Tubería (pipe). Permite la comunicación entre programas.
- s : Socket. Similar al pipe pero permite enlaces de red bidireccionales.
- - : Fichero regular (archivo normal y corriente).

- 2º 3º 4º carácter. Es el primer trío de permisos, y nos indican los permisos que el usuario propietario (U) tiene sobre ese fichero. Este usuario es el que aparece en la línea del `ls`.
- 5º 6º 7º carácter. Es el segundo trío de permisos, y nos indica los permisos que el grupo propietario (G) tiene sobre ese fichero. Este grupo es el que aparece en la línea del `ls`.
- 8º 9º 10º carácter. Es el tercer trío de permisos, y nos indica los permisos que los otros (O) tienen sobre ese fichero. Otros se refiere a cualquier usuario que no sea el usuario propietario del fichero (U) ni pertenezca al grupo propietario del fichero (G). Se le suele denominar *permisos globales*.

| **TIPO** | **USUARIO (U)** | | | **GRUPO (G)** | | | **OTROS (O)** | | |
|---|---|---|---|---|---|---|---|---|---|
| **-** | **r** | **w** | **x** | **r** | **w** | **x** | **r** | **w** | **x** |
| | READ | WRITE | EXEC | READ | WRITE | EXEC | READ | WRITE | EXEC |
| | Leer | Escribir | Ejecutar | Leer | Escribir | Ejecutar | Leer | Escribir | Ejecutar |

Para cada uno de estos tres tríos vemos que existen tres tipos de permisos fundamentales:

- **r**: read (lectura). El usuario que tenga este permiso podrá, si es un directorio listar los recursos almacenados en él, y si es cualquier otro tipo de fichero podrá leer su contenido.
- **w**: write (escritura). Todo usuario que posea este permiso para un fichero podrá modificarlo. Si se posee para un directorio se podrán crear y borrar ficheros en su interior.
- **x**: execute (ejecución). Este permiso para el caso de los ficheros permitirá ejecutarlos desde la línea de comandos y para los directorios, el usuario que lo

posea tendrá acceso para realizar el resto de las funciones permitidas mediante los otros permisos (lectura y/o escritura). **Si un usuario no tiene permiso de ejecución en un directorio, directamente no podrá entrar en el mismo, ni pasar por él.**

Los tres tipos de permisos mencionados poseen una representación numérica basada en el sistema octal que parte de representar como "1" los bits de los permisos otorgados y "0" para los denegados. Luego se transforma la representación binaria así obtenida en octal. De esta forma se obtienen para cada tipo de permiso los siguientes valores:

| Permiso | R | W | X |
|---|---|---|---|
| Valor binario | 4 | 2 | 1 |

La combinación de los tres tipos de permisos para un tipo de usuario oscila desde cero (ningún permiso) hasta siete (todos los permisos).
Ejemplos:

**rw- = 110 (4 + 2 + 0) (6 en octal)**
**rwx = 111 (4 + 2 + 1) (7 en octal)**
**r-x = 101 (4 + 0 + 1) (5 en octal)**
**r-- = 100 (4 + 0 + 0) (4 en octal)**

Así, los permisos "totales" de un recurso constan de nueve indicadores, donde los tres primeros indican los permisos asociados al propietario, los otros tres al grupo propietario y los últimos al resto de los usuarios.

Ejemplos:

**rwx r-x --- = 111 101 000 (750 en octal)**
**rw- r-- r-- = 110 100 100 (644 en octal)**

Sólo el propietario de un recurso siempre tendrá derecho a cambiar sus permisos, además del usuario **root**, por supuesto.

Existen otros tipos de permisos más complejos (sólo los vemos a nivel informativo):

- s y S: es un permiso que de no administrarse correctamente puede provocar problemas de seguridad. Para su representación a través de caracteres se utiliza el lugar del permiso de ejecución y de ahí la diferencia entre s y S: si es *s* (minúscula) significa que incluye además el permiso de ejecución a diferencia de *S* (mayúscula). Este permiso se puede asociar al dueño o al grupo del recurso. Si se asocia a un fichero significa que cuando este se ejecute por un usuario que tenga permisos para ello adquirirá los permisos de su dueño o grupo en dependencia de a cuál de los dos está asociado el permiso. Un ejemplo de fichero con este permiso es el comando *passwd*, el cual adquiere los permisos de *root* al ser ejecutado por los usuarios (sin argumentos) para poder modificar el fichero */etc/shadow* que es donde se guardan las contraseñas de los usuarios. Para el caso de un directorio este permiso sólo tiene validez para el grupo del mismo permitiendo a los ficheros y a los subdirectorios que se creen en él heredar el grupo, los subdirectorios heredarán también el permiso *s*. Un ejemplo de directorio con este permiso es aquel donde se guardan los documentos de un sitio FTP anómimo. Este permiso se conoce como **setuid bit** o **setgid bit**, para el usuario y el grupo, respectivamente.

- t y T: cuando está asociado a un directorio junto al permiso de escritura para un grupo de usuarios, indica que estos usuarios pueden escribir nuevos ficheros en el directorio pero estos sólo podrán ser borrados por sus dueños o por *root*. Para un fichero el permiso expresa que el texto de este se almacena en memoria swap para ser accedido con mayor rapidez. Este permiso sólo se asocia al resto de los usuarios y para su representación se emplea el bit correspondiente al permiso de ejecución: si es *t* (minúscula) significa que incluye además el permiso de ejecución y *T* (mayúscula) no lo incluye. Ejemplo de un directorio con este permiso es */tmp* donde todos los usuarios pueden escribir pero sólo los dueños pueden borrar sus ficheros, además de *root*. Este permiso se conoce también como **sticky bit**.

Para representar los permisos *t* y *s* en el sistema binario se utilizan tres bits adicionales: el primero para *s* en el dueño, el segundo para s en el grupo y el tercero para *t*. Estos se colocan al inicio de la cadena numérica de nueve bits vista anteriormente. En la cadena

de caracteres se mezclan con el permiso de ejecución y de ahí la necesidad de emplear las mayúsculas y minúsculas.

Ejemplos:

**rws rwS r-- = 110 111 110 100 (6764 en octal)**

**rwx rws -wT = 011 111 111 010 (3772 en octal)**

Vemos como el primer trío de bits, nos indica si deseamos activar los permisos s para el usuario, los permisos s para el grupo, y los permisos t para el resto. Los otros tres tríos de bits son los permisos tal como los hemos visto anteriormente.

**Comando chmod**

Para cambiar los permisos de un recurso se utiliza el comando *chmod*.

Sintaxis: chmod [opciones] <permisos> <ficheros>

Las formas de expresar los nuevos permisos son diversas, se puede emplear la representación numérica o utilizando caracteres.

Utilizando caracteres, la orden *chmod* se usa así:

**Chmod (letra del trío a cambiar) (+ - =) (permisos) fichero**

Donde:

- Letra de trío a cambiar puede ser:
  - U - Usuario (1º trío)
  - G - Grupo (2º trío)
  - O - Otros (3º trío)
  - A - All, todos los tríos (U, G y O).
- El carácter puede ser:

  + Otorga el permiso

  – Quita el permiso

  = Deja el permiso exactamente igual a lo que se indica (es decir, quitará todos los demás).
- Permisos puede ser cualquier combinación de:
  - r para leer

- w para escribir
- x para ejecutar

Utilizando la representación numérica, simplemente hay que poner:

**chmod (representación numérica) fichero**

Ejemplos:

| | |
|---|---|
| `$ chmod u+x clase.txt` | `# añade el permiso de ejecución al dueño` |
| `$ chmod g=rx program.sh` | `# asigna exactamente los permisos de lectura y ejecución al grupo` |
| `$ chmod go-w profile` | `# elimina el permiso de escritura en el grupo y en otros` |
| `$ chmod a+r,o-x *.ts` | `# adiciona el permiso de lectura para todos los usuarios y elimina el de ejecución para otros` |
| `$ chmod +t tmp/` | `# adiciona el permiso especial t` |
| `$ chmod 755 /home/pepe/dc/` | `# asigna los permisos con representación octal 755 (rwx r-x r-x)` |
| `$ chmod -R o+r apps/` | `# adiciona el permiso de lectura a otros para un directorio de forma recursiva (incluyendo todo su contenido)` |
| `$ chmod +x ./bin/*` | `# adiciona el permiso de ejecución a todos los usuarios que les corresponde por defecto` |
| `# chmod 4511 /usr/bin/passwd` | `# asigna los permisos con representación octal 4511 (r-s--x--x)` |

## Comando umask

Para determinar qué permisos se asocian por defecto a los ficheros o directorios creados, cada usuario posee una máscara de permisos. Esta se expresa en el formato numérico octal, es decir, posee tres dígitos entre cero y siete (Ej. 166). La máscara indica que permisos **no** se desea que tenga el recurso creado. Por defecto esta máscara es 002 para los usuarios comunes y 022 para root. La máscara realmente se asocia al shell y se hereda por los subshells.

Para calcular los permisos finales que se obtienen con la máscara 022 se hace la siguiente operación por parte del sistema:

Ficheros = totales_para_ficheros – máscara = 666 - 022 = 644 = -rw-r--r--
Directorios = totales_para_directorios – máscara = 777 - 022 = 755 = drwxr-xr-x

Para fijar o visualizar la máscara se puede emplear el comando *umask*.
Sintaxis: umask [-S] [máscara]

Ejemplos:

```
$ umask           # sin argumentos muestra la máscara actual en formato
                  numérico
$ umask -S        # muestra el complemento de la máscara en formato de
                  caracteres u
$ umask 037       # asigna la máscara 037 (niega permisos de ejecución
                        y de escritura para el grupo, y todos los
                  permisos para el resto de los usuarios).
$ umask g=rx,o=   # especifica el complemento la máscara utilizando el
                  formato de caracteres
```

Para ser bien restrictivos se recomienda hacer:

```
$ umask 077
```

Los nuevos directorios tendrán el permiso: 700 = drwx------
Los nuevos ficheros tendrán el permiso: 600 = -rw-------

Resumiendo, si queremos que a partir de este momento los nuevos ficheros que creemos, tengan como permiso por ejemplo rw- r-- ---, calculamos la representación numérica de estos permisos, que en este caso son 640. Ahora restamos a 666 el número obtenido, y obtenemos 026, pues este es el número que debemos usar en umask para conseguir lo que queremos.

```
# umask 026
```

Si en lugar de pensar en ficheros, pensamos en directorios, habría que restar no 666, sino 777. (Se recomienda no trabajar en números impares con *umask*, es decir, obviamos trabajar con el bit del permiso de ejecución).

Las distribuciones Linux normalmente tienen una máscara por defecto de 002 o 022.

## Comando chown

El comando *chown* se utiliza para cambiar el dueño y el grupo de un fichero. Existe también el comando *chgrp* que se emplea de forma similar pero para cambiar el grupo solamente. El dueño de un fichero solo lo puede cambiar el usuario *root* mientras que el grupo además de *root*, lo puede cambiar el propio dueño, siempre que pertenezca al nuevo grupo.

El propietario y el grupo de un fichero se puede comprobar, como ya hemos visto, mediante el mandato `ls -l`

Sintaxis:

chown [opciones] <dueño>[.grupo] <ficheros>

chown [opciones] <grupo> <ficheros>

Opción:

-R en los directorios cambia el dueño y/o el grupo recursivamente.

Ejemplos:

```
# chown pepe.pepe tesis/      # cambia el propietario de tesis a pepe y
                              el grupo de tesis al grupo pepe
# chown -R root /tmp/oculto   # cambia todos los ficheros que estén en
                              el directorio oculto y coloca como
                              propietario al usuario root

# chgrp ftpusers /usr/ftp     # cambia el grupo propietario del fichero
                              /usr/ftp al grupo ftpusers
# chown .ftpusers /usr/ftp    # lo mismo que el mandato anterior
```

El modelo que hemos estudiado de permisos se ha empleado desde la creación de Linux. Se denomina DAC (Control de Acceso Discrecional). En el sector de la seguridad se considera insuficiente. Existen otros modelos como ACL (comandos setfacl o getfacl), MAC y RBAC (utilidad SELinux). No los estudiamos.

# 5. Perfil de seguridad y política de contraseñas

## *Permisos de la carpeta de usuario*

Como hemos visto, cuando se crea un nuevo usuario se crea un directorio nuevo llamado */home/usuario* que es lo que conocemos como carpeta personal. El perfil predeterminado sobre el que se crea esta carpeta es el modelo de los contenidos que se encuentran en el directorio */etc/skel*, que incluye todos los elementos básicos del perfil.

Si ejecutamos el mandato *useradd* con la opción de mostrar valores por defecto, veremos que efectivamente el *esqueleto* de la carpeta personal lo tomará del directorio /etc/skel:

```
$ useradd -D
GROUP=100
HOME=/home
INACTIVE=-1
EXPIRE=
SHELL=/bin/sh
SKEL=/etc/skel
CREATE_MAIL_SPOOL=no
```

Los valores por defecto del mandato ***useradd*** están a su vez almacenados en un fichero dentro del directorio */etc/default*. Este fichero es el */etc/default/useradd*. En él podemos cambiar ciertos parámetros por defecto como la ubicación de la carpeta personal, el Shell asignado o el esqueleto de la carpeta personal.

Si por el contrario utilizamos el mandato ***adduser*** no tiene en cuenta este fichero de valores por defecto toma los suyos propios del fichero */etc/adduser.conf*

Otro fichero importante que controla ciertos comportamientos por defecto de los comandos *useradd* y *adduser* es el fichero */etc/login.defs*

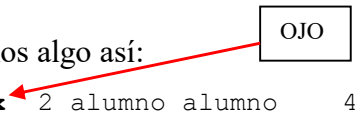
Dependiendo del comando a utilizar, y de la distribución y versión de Linux, los valores por defecto de la creación de usuarios los cogerá de uno u otro de estos ficheros.

Pero además, el comprtamiento de la creación del directorio *esqueleto* que hemos visto funciona para usuarios que abren sesión de terminal y no desde la interfaz gráfica. Si el usuario abriera sesión desde el entorno gráfico, en su primer inicio de sesión se crearía otra estructura de carpetas (además de la ya existente) en su carpeta personal (incluiría los directorios *Escritorio, Documentos, Descargas,* etc …). Esta nueva estructura la toma (junto con otras informaciones) de la carpeta */etc/xdg*, concretamente de los ficheros */etc/xdg/user-dirs.defaults* y */etc/xdg/user-dirs.conf*

Hay que poner especial atención en los permisos de los directorios personales para garantizar la confidencialidad. De forma predeterminada, los directorios personales de cada usuario de Ubuntu se crean con permisos de lectura y ejecución para todo el mundo. Esto significa que todos los usuarios pueden entrar y acceder al contenido del directorio personal de otros usuarios. Esto puede que no sea apropiado para nuestro entorno. Para verificar los permisos de nuestra carpeta personal tecleamos:

```
$ ls -ld /home/alumno
```

Mostrándonos algo así:




```
drwxr-xr-x  2 alumno alumno    4096 2007-10-02 20:03 alumno
```

Para cambiar estos permisos de la carpeta ya creada:

```
$ sudo chmod 0750 /home/alumno
```

No es necesario usar la opción recursiva (-R), lo cual modifica todas las carpetas y los archivos hijos. Además de no ser necesario puede dar lugar a resultados no deseados. El directorio padre, por sí solo, es suficiente para impedir el acceso no autorizado a cualquier cosa que haya dentro del padre.

Pero una aproximación mucho más eficiente al asunto podría ser modificar los permisos globales predeterminados cuando se crean las carpetas personales de los usuarios mediante los comandos de creación de usuario.

Para ello, dependiendo del comando a utilizar para la creación y de la distribución de Linux que usemos, editaremos el archivo */etc/adduser.conf* y modificamos la variable DIR_MODE a algo más apropiado, de forma que todos los nuevos directorios personales recibirán los permisos correctos (DIR_MODE=0750); o bien en el archivo */etc/login.defs* poniendo la variable UMASK 027.

## Políticas de contraseñas

Una política de contraseñas rigurosa es uno de los aspectos más importantes en cuanto a la seguridad. Muchas violaciones exitosas de seguridad son posibles simplemente mediante ataques de fuerza bruta y diccionarios frente a contraseñas débiles. Hemos de comprobar que tenemos configurados de manera adecuada los requisitos mínimos de complejidad de las contraseñas y los tiempos de vida máximo de las contraseñas, y que llevamos a cabo auditorías frecuentes de los sistemas de autenticación.

Ya hemos visto sin entrar en detalle el comando *chage* utilizado para cambiar los valores de caducidad de la contraseña.

Cuando se crean cuentas de usuario, se debería seguir una política de tiempo de vida mínimo y máximo para las contraseñas, que fuerce a los usuarios a cambiar sus contraseñas cuando estas expiren.

Para ver de manera sencilla el estatus actual de una cuenta de usuario tecleamos:

```
$ sudo chage -l alumno
```

La salida siguiente muestra algunos hechos interesantes sobre la cuenta del usuario, con la premisa de que no se ha aplicado ninguna política:

```
Último cambio de contraseña                          : abr 05, 2016
La contraseña caduca                                 : nunca
Contraseña inactiva                                  : nunca
La cuenta caduca                                     : nunca
Número de días mínimo entre cambio de contraseña     : 0
Número de días máximo entre cambio de contraseña     : 99999
Número de días de aviso antes de que caduque la contraseña : 7
```

Para establecer alguno de esos valores utilizamos el mandato *chage*:

```
$ sudo chage alumno
```

Otro aspecto importante es la longitud de la contraseña y su complejidad. De manera predeterminada, Ubuntu requiere que la contraseña tenga una longitud mínima de 6 caracteres, así como algunas comprobaciones básicas de entropía. Estos valores se controlan en el archivo ***/etc/pam.d/common-password***, que se describe a continuación.

```
password      [success=1 default=ignore]    pam_unix.so obscure sha512
```

Si queremos ajustar la longitud mínima a 8 caracteres, cambiamos la variable apropiada a min=8:

```
password [success=1 default=ignore]pam_unix.so obscure sha512 minlen=8
```

Para prevenir el uso de contraseñas utilizados hacemos uso del parámetro *remember* en la línea de password que tiene a "pam_unix.so". Si le damos, por ejemplo, un valor de 5, no permitirá la reutilización de las últimas cinco contraseñas almacenadas en */etc/security/opasswd*.

```
password [success=1 default=ignore] pam_unix.so obscure sha512
remember=5
```

En este fichero podemos también configurar la obligación de usar mayúsculas, minúsculas, números y caracteres especiales en la contraseña.

# 6. Gestión de procesos

Un proceso es una instancia de un programa en ejecución. Un programa almacenado en disco, cuando lo ejecutamos, se convierte en un proceso, se carga en memoria y comienza a ejecutarse en el procesador.

En Linux se ejecutan muchos procesos de forma concurrente aunque realmente sólo uno accede al procesador en un instante de tiempo determinado. Esto en esencia es lo que caracteriza a los sistemas multitarea.

Cada proceso en el momento de su creación se le asocia un número único que lo identifica del resto. Además a un proceso están asociadas otras informaciones tales como:

- El usuario que lo ejecuta.
- La hora en que comenzó.
- La línea de comandos asociada.
- Un estado. Ejemplos: *sleep*, *running*, *zombie*, *stopped*, etc.

- Una prioridad que indica la facilidad del proceso para acceder a la CPU. Oscila entre -20 y 19, donde -20 es la mayor prioridad.
- La terminal donde fue invocado, para el caso de que este asociado a alguna terminal.

## Comando ps

Para ver los procesos y sus características se emplea el comando **ps**. Una salida típica de este comando es:

```
# ps
PID   TTY          TIME        CMD
1035  pts/0        00:00:14    bash
1831  pts/0        00:00:00    ps
```

Como puede apreciarse para cada proceso se muestra su ID (identificación o número), el terminal donde se invocó, el tiempo de CPU que se le ha asignado hasta el momento y el comando que lo desencadenó. Por defecto *ps* muestra en formato reducido los procesos propios del usuario y el terminal actual.

Algunas opciones de ps.

x : muestra todos los procesos del usuario actual sin distinción de terminal.

a : muestra todos los procesos de todos los usuarios.

f : muestra las relaciones jerárquicas entre los procesos.

e : muestra el entorno de cada proceso.

l : utiliza un formato más largo (muestra más información).

u : utiliza un formato orientado a usuario.

Ejemplos:

```
$ ps aux
$ ps e
$ ps xf
```

## Comando top

Otro comando para ver el estado de los procesos en ejecución es **top**, que permite hacerlo dinámicamente. El comando top es más bien un programa interactivo con el cual se pueden observar los procesos más consumidores de CPU por defecto. Es una

buena herramienta para localizar procesos descontrolados, con mucho consumo de recursos.

Este comportamiento se puede modificar tecleando:
m : ordenará según el empleo de la memoria.
p : ordenará según el empleo de la CPU.
n : ordenará por ID.
a : ordenará por antigüedad.
k : Podemos matar un proceso con este comando indicando su PID

Para observar todos los posibles comandos durante la interacción con top se puede pulsar h. Para salir se presiona q.

El programa top muestra además algunas estadísticas generales acerca del sistema:

- La hora actual y en la que se inició el sistema.
- La cantidad de usuarios conectados.
- Los promedios de carga de la CPU en los últimos uno, cinco y quince minutos transcurridos.
- Un resumen estadístico de la cantidad de procesos en ejecución y su estado (sleeping, running, zombie y stopped).
- Un resumen del empleo de la memoria física y virtual (swap).
- Los tres primeros aspectos se pueden ver también con el comando uptime y el último con free.

**Control de los procesos en bash: & fg bg**

El shell bash permite ejecutar los procesos en *foreground* (primer plano) o *background* (segundo plano). Los primeros son únicos por terminal (no puede haber más de un proceso en primer plano en cada terminal) y es la forma en que se ejecuta un proceso por defecto. Solo se retorna al prompt una vez que el proceso termine, sea interrumpido o detenido.

En cambio, en *background* pueden ejecutarse muchos procesos a la vez asociados al mismo terminal. Estos se ejecutan en segundo plano y no deben interactuar con el usuario (entrada y/o salida por el terminal). Una vez lanzado, el terminal nos muestra el

prompt para seguir ejecutando mandatos sin esperar a que el que se ha lanzado en segundo plano termine.

Para indicar al shell que un proceso se ejecute en *background*, se utiliza un símbolo *ampersand* (&) al final de la línea.

Ejemplos:
El siguiente mandato puede costarle un tiempo considerable al sistema:

```
$ sudo ls / -R > contenido_de_disco.txt
```

Podríamos lanzarlo en segundo plano (*background*) y continuar trabajando sin esperar a que terminara:

```
$ sudo ls / -R > contenido_de_disco.txt &
```

Otro mandato costoso:

```
# updatedb&
```

Para modificar el estado de un proceso en *foreground* desde bash existen dos combinaciones de teclas muy importantes que este interpreta:

- Ctrl+C : trata de interrumpir el proceso en *foreground*. Si es efectivo, el proceso finaliza su ejecución (es asesinado).
- Ctrl+Z: trata de detener el proceso en *foreground*. Si es efectivo el proceso continúa activo aunque deja de acceder al procesador (está detenido y pasa a *background*).

Para ver los procesos detenidos o en *background* en un shell se emplea el comando integrado en el bash **jobs**, que mostrará una lista con todos los procesos en dichos estados mediante los comandos asociados y un identificador numérico especial.
Ejemplo:

```
# jobs
[1]   Running   sleep 10000 &
[2]-  Stopped   cp /var/log/messages /tmp
[3]+  Stopped   updatedb
```

Los procesos detenidos se pueden llevar al *background* y estos a su vez pueden trasladarse al *foreground*. Para ello se emplean respectivamente los comandos integrados al bash: **bg** y **fg**, pasándoles como argumento el identificador especial del proceso. Si no se especifica un argumento se asumirá el trabajo marcado con un signo "+" que sería el último detenido o llevado al *background*.

Ejemplos:

```
$ bg 2
[2]- cp /var/log/messages /tmp &

$ fg
upadtedb
```

Si se comienza a ejecutar un proceso y este se demora mucho y no interesa por el momento sus resultados se puede detener y enviarlo al *background* haciendo **Ctrl+Z** y luego, ***bg***.

## Comunicación con los procesos: kill

Un comando muy útil para interactuar con los procesos es *kill*. Este permite enviarles señales con significados muy diversos. Los programas o comandos deben estar preparados para atrapar y tratar estas señales, al menos las más importantes. Existen muchos tipos de señales, para verlas se puede escribir en Linux `kill -l`.

```
$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL
 5) SIGTRAP      6) SIGABRT      7) SIGBUS       8) SIGFPE
 9) SIGKILL     10) SIGUSR1     11) SIGSEGV     12) SIGUSR2
13) SIGPIPE     14) SIGALRM     15) SIGTERM     17) SIGCHLD
18) SIGCONT     19) SIGSTOP     20) SIGTSTP     21) SIGTTIN
22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
```

Por defecto *kill* envía la señal *TERM* que indica al proceso que debe terminar (15). La señal 9 o *KILL* lo finaliza forzosamente (es como una orden *TERM* pero imperativa). La señal *HUP* es interpretada por muchos comandos y programas como una indicación de que relean los ficheros de configuración correspondientes (que reinicien su ejecución).

Así, al ejecutar *kill*, hay que indicar qué señal se quiere mandar, y a qué número de proceso se le quiere mandar dicha señal.

Ejemplos:

```
$ kill 1000             # envía la señal 15 (TERM) al proceso 1000
$ kill -s 9 10101       # envía la señal 9 (KILL) al proceso 10101
$ kill -4 18181         # envía la señal 4 (ILL) al proceso 18181
# kill -HUP 199         # envía la señal 1 (HUP) al proceso 199
$ kill %2               # envía la señal 15 (TERM) al trabajo 2 (en
                  background o detenido)
```

También existe **killall** que permite enviar señales a los procesos a través de sus nombres. A diferencia del ID, el nombre de un proceso no es único, o sea pueden existir muchos procesos con el mismo nombre y de ahí la utilidad de este comando.

Sintaxis: killall [opciones] [-señal] <nombre>
Algunas opciones:
-i : forma interactiva. Pregunta para cada proceso si se desea enviar la señal o no.
-v : reporta si tuvo exito el envío de la señal.

Ejemplo:

```
$ killall -9 gdm
```

# 7. Otros comandos

**Comando exit**

El comando *exit* permite terminar el shell actual. Si se tiene un único shell es equivalente a desconectarse del sistema, pero si se está en un subshell sólo se terminará este, retornando al shell anterior.

**Comando logout**

El comando *logout* permite desconectarse del sistema a partir de un login shell (primer shell que se ejecuta al establecer la conexión).

La secuencia de caracteres Ctrl+d permite terminar el shell actual. Si es un login shell equivaldrá a hacer *logout* y si no, a hacer exit.

**Comando halt, reboot, poweroff**

Estos comandos nos permiten suspender, reiniciar o apagar el sistema.

**Comando shutdown**

Este comando nos permite "echar abajo" el sistema.

Algunas opciones interesantes:

-c cancela un *shutdown* que está en proceso de ejecución

-f Reinicia más rápido, ya que no controla la integridad de los sistemas de archivos

-h Cuando el sistema se apaga, apaga el ordenador (o lo intenta)

-r Cuando el sistema se apaga, intenta reiniciarlo

Después de estas opciones, se le indica cuando queremos apagar el sistema:

now lo apaga inmediatamente, ahora.

20:00 lo apaga a las 8 de la tarde

+10 lo apaga en 10 minutos

```
# shutdown -h +4
```

**Comando who**

El comando *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 qué terminal lo han hecho y en qué 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.

Ejemplos:

```
$ who
$ who am I
```

**Comando w**

El comando *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, cuánto tiempo lleva funcionando, cuantos usuarios hay conectados (sin incluir las sesiones gráficas) y tres porcentajes de carga de la CPU: durante el último minuto, los 5 y los 15 minutos anteriores. A continuación se muestra una tabla cuyas columnas representan: el login de cada usuario conectado, por qué terminal lo ha hecho, desde qué host, a qué 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.
Ejemplos:

```
$ w
$ w pepe
```

**Comando echo**

El comando *echo* muestra en su salida todo lo que se le pase como argumento.
Ejemplo:

```
$ echo Hola mundo...
Hola mundo...
```

## Comando set

El comando *set* sirve para gestionar las variables y funciones del shell. Si se ejecuta set directamente sin parámetros, nos devuelve los nombres y valores de varias funciones del shell.

```
# set
```

En bash no es necesario asignar variables mediante *set variable=valor*, sino que se puede usar directamente *variable=valor*. Sin embargo, nos podemos encontrar trabajando con otros shells, como *csh* en los que es obligatorio usar el set para asignar valores a las variables.

## Comando history

El comando *history* nos permite controlar el histórico de comandos que se van almacenando en el shell. El histórico se almacena por defecto en el fichero ~/.history lo que nos indica que existe un histórico diferenciado para cada usuario. Si ejecutamos el comando *history* nos devolverá la relación de comandos para ese usuario.

```
# history
```

Una posibilidad es enviar la salida de la orden *history* a un fichero, con lo que conseguimos tener una relación de los comandos que hemos ejecutado.

## Comando uname

El comando *uname* nos da información sobre el sistema, como el nombre del kernel, su versión, el nombre de la máquina, etc.

```
# uname -a              # Muestra toda la información
# uname -r              # Muestra la release del kernel
```

## Comando date

El comando *date* sirve para consultar y cambiar la fecha y hora del sistema.

```
# date
```

El comando *date* tiene bastantes opciones, y con él se pueden hacer bastantes cosas. Es recomendable echarle un vistazo a sus páginas de manual para entenderlos bien. Algunos ejemplos:

```
$ date --date="2 days ago"          # nos da la fecha de hace dos días
$ date --date="3 months 1 day"      # nos da la fecha que será en 3
                                    meses y un día.
$ date --date="25 Dec"              # nos da el dia que será el 25 de
                                    Diciembre del año actual
$ date "+%B %d"                     # nos devuelve el nombre del mes
                                    actual y el número de día del mes.
$ date --set="2006-6-20 11:59 AM"   # ajusta la fecha del sistema
$ date --set="+3 minutes"           # adelante la hora del sistema en 3
                                    minutos.
```

## Comando time

El comando *time* sirve para cronometrar cuánto tiempo tarda en ejecutarse un comando cualquiera, y qué recursos consume. Puede ser una orden interesante para comprobar el rendimiento de los sistemas.

```
# time wc /etc/passwd
```

## Comando uptime

El comando *uptime* nos indica las el "uptime" del sistema. En informática se conoce como *uptime* el tiempo que un equipo lleva "levantado", es decir cuanto tiempo lleva el sistema funcionando. En un servidor es muy importante que este *uptime* sea lo más elevado posible. *Uptime* nos devuelve cuánto tiempo lleva el equipo levantado, el número de usuarios que están conectados, y la media de las cargas de sistema para los últimos 1, 5, y 15 minutos.

```
# uptime
```

## Comando dmesg

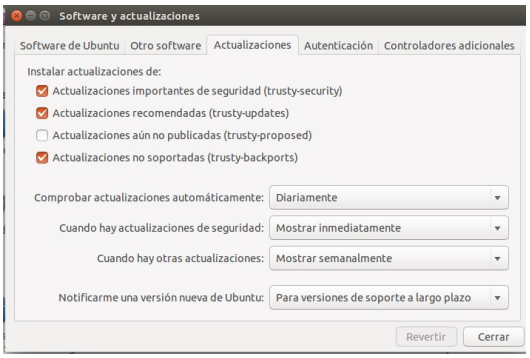
Nos da una lista con todos los mensajes que se han producido durante el arranque del sistema.

```
# dmesg
# dmesg | tail -5
```

# 8. Actualización del sistema operativo e instalación de software

En todos los SO es necesario tener actualizado el sistema debido a que continuamente se están generando nuevos paquetes que, o bien solucionan un error detectado, o bien añaden nuevas funcionalidades al sistema operativo. En cualquiera de los 2 casos, si queremos que nuestro equipo esté a la última deberemos actualizar el sistema continuamente.

Para ello, todos los SO tienen un gestor de actualizaciones que se encarga de comprobar frecuentemente si existe una nueva actualización, permitiendo instalarla automáticamente o avisar al usuario para que lo haga cuando lo considere oportuno. También nos da la opción, poco recomendable, de no comprobar si existen actualizaciones.


En la distribución concreta de Ubuntu, se dispone de un gestor de actualizaciones desde el que se pueden instalar las actualizaciones disponibles y también una aplicación donde se pueden configurar las comprobaciones automáticas y los repositorios de descarga de software, ubicado en el menú **Configuración del Sistema → Software y Actualizaciones**. Podemos acceder a ambas aplicaciones desde el tablero:


Además de las actualizaciones del sistema operativo, también es posible en Linux instalar software adicional al que viene instalado. Se utiliza para ello los denominados *paquetes de software* que son una serie de programas que se distribuyen conjuntamente encapsulados en un solo fichero. Los sistemas operativos Linux incluyen los **Gestores de Paquetes** para administrar estos paquetes de software. Dependiendo de la distribución se utilizan unos gestores u otros. Los principales gestores de paquetes son:

- *dpkg:* Paquetes Debian (paquetes *.deb*). Se incluye este gestor de paquetes en distribuciones como Ubuntu, Mint, Xandros, y por supuesto Debian.
- *RPM*: Paquetes RPM (Red Hat. Paquetes *.rpm*). Quizás el más popular. Es el que utilizan distribuciones como Suse, Fedora, CentOS y RHEL.
- *tgz*: Es un archivo de paquetes específico para Unix comprimido. Es un paquete que contiene las aplicaciones y su código fuente, para no tener que crear un tipo de paquete específico para cada distribución. A diferencia de los paquetes .deb, o .rpm, este no contiene instrucciones particulares de instalación para cada distribución, por lo que la instalación del contenido deberá ser compilado por el usuario.

El sistema de gestión de paquetes de Ubuntu está derivado del mismo sistema utilizado por la distribución Debian. Los paquetes contienen todos los archivos necesarios, meta-datos e instrucciones para implementar una funcionalidad particular o una aplicación software en un equipo Ubuntu.

Los archivos de paquetes Debian normalmente tienen la extensión ".deb", y por lo general existen en **repositorios** que son colecciones de paquetes que se encuentran en

diversos medios, tales como discos CD-ROM, o por supuesto, en repositorios en línea. Los paquetes están normalmente en un formato binario pre-compilado; de este modo la instalación es rápida, y no requiere compilación de software.

Muchos paquetes complejos utilizan dependencias. Las dependencias son paquetes adicionales requeridos por el paquete principal para que funcione correctamente. Las herramientas de administración de software en Ubuntu instalan estas dependencias de forma automática.

En Ubuntu es posible el añadir, eliminar o actualizar aplicaciones (mediante estos paquetes) de varios modos que vamos a ver en detalle.

## *Centro de software de Ubuntu*

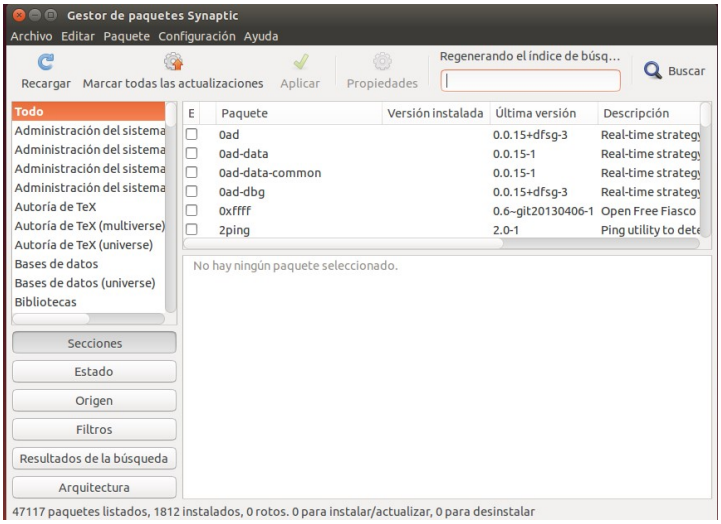
El centro de software de Ubuntu es una sencilla aplicación que se encuentra en el lanzador con la que se puede añadir o quitar paquetes del sistema de una manera muy sencilla mediante una interfaz gráfica. Es la manera más sencilla de hacerlo.


Basta con buscar la aplicación que deseemos seleccionar la opción de instalar. También se nos permite desinstalar software ya instalado.

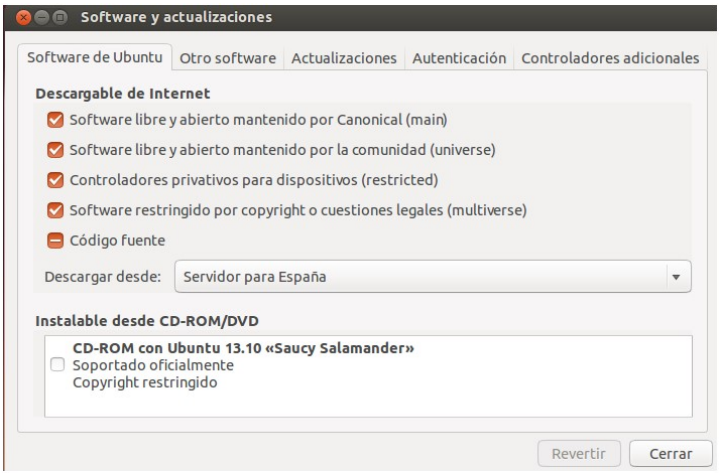
## Synaptic

*Synaptic* es una aplicación gráfica para *GNOME* con la que también se pueden instalar paquetes, pero esta vez con un mayor control sobre los programas que se instalan en el sistema, así como un mayor número de ellos. *Synaptic* es el equivalente gráfico a la instrucción ***apt-get*** que veremos a continuación, de hecho lo utiliza de manera interna. En las versiones de Ubuntu con *Unity* no viene instalado por defecto *Synaptic*, hay que instalarlo desde el *Centro de Software*.

Funciona de manera parecida al centro de software, pero aquí vemos directamente los paquetes y sus dependencias.




Desde la ventana de configuración del software y aplicaciones (que hemos visto anteriormente y a la que podemos acceder desde el propio *Synaptic*) se le puede indicar los repositorios (almacenes de software) desde donde descargar los paquetes.




El significado de esta pantalla lo veremos a continuación con el *apt-get*.

## Adept

El programa *Adept* es la versión de *Synaptic* para KDE. Viene incluida en Kubuntu.

## dpkg

Como hemos visto, *dpkg* es un gestor de paquetes para sistemas basados en Debian. Puesto que Ubuntu desciende de Debian, se utiliza el mismo gestor de paquetes. Se puede instalar, eliminar, modificar y generar paquetes, pero a diferencia de otros sistemas de gestión de paquetes, no se puede descargar e instalar automáticamente los paquetes o sus dependencias.

Los ficheros con extensión *.deb* son paquetes de aplicaciones ya preparados para instalarse de una forma sencilla. Si los descargamos desde un navegador mediante la interfaz gráfica, Ubuntu los reconoce en el momento de descargarlos y ejecutarlos, y automáticamente lanzará la aplicación ***gdebi***, que se ocupará de instalar el paquete y buscar las dependencias de otros paquetes que pudiera necesitar para su correcta instalación.

También se puede 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>
```

Podemos visualizar todos los paquetes instalados en el sistema mediante 2 mandatos:

```
$ dpkg -l
$ dpkg --get-selections
```

Supongamos que deseamos instalar el paquete samba_4.1.6+dfsg-1ubuntu2.1404.3_amd64.deb y para ello necesitamos desinstalar una versión previa:

```
$ sudo dpkg -r samba
$ sudo dpkg -i samba_4.1.6+dfsg-1ubuntu2.1404.3_amd64.deb
```

## apt-get

El comando *apt -get* es una potente herramienta de línea de comandos, que trabaja con Advanced Packaging Tool de Ubuntu (APT) para llevar a cabo funciones tales como la instalación de nuevos paquetes, actualización de paquetes existentes, la actualización del índice de paquetes, e incluso la actualización de todo el sistema Ubuntu. Es una herramienta de meta-empaquetado.

Al ser una sencilla herramienta de línea de comandos, *apt- get* tiene numerosas ventajas sobre otras herramientas de gestión de paquetes disponibles en Ubuntu para los administradores. Algunas de estas ventajas incluyen la facilidad de uso a través de conexiones de terminales simples (SSH), y la capacidad para ser utilizado en las secuencias de comandos de administración del sistema, que a su vez puede ser automatizado por la utilidad de programación de tareas *cron*.

Para instalar un paquete con *apt-get* ejecutaríamos el siguiente comando con la opción *install*:

```
$ sudo apt-get install <paquetes>
```

Por ejemplo:

```
$ sudo apt-get install nmap
```

Para la instalación, el programa *apt-get* comprueba en la **lista de repositorios** de dónde debe descargarse el paquete en cuestión, lo descarga y lo instala por nosotros en el sistema. Todo de manera automática.

Para desinstalar paquetes utilizaríamos la opción *remove*:

```
$ sudo apt-get remove <paquetes>
```

Por ejemplo:

```
$ sudo apt-get remove nmap
```

Añadiendo la opción *--purge* a *apt-get remove* también se eliminarán los archivos de configuración del paquete. Esto puede o no tener el efecto deseado, así que hay que usarlo con precaución.

Desinstalar paquetes (incluyendo archivos de configuración):

```
$ sudo apt-get purge <paquetes>
```

Se pueden especificar varios paquetes para instalar o desinstalar, separándolos por espacios.

Con el tiempo, los repositorios irán añadiendo versiones actualizadas de los paquetes instalados en nuestro equipo (por ejemplo, actualizaciones de seguridad). Para actualizar el sistema con las actualizaciones de paquetes disponibles hay que ejecutar:

```
$ sudo apt-get upgrade
```

Por último, **es necesario tener actualizada la lista de los paquetes disponibles y sus versiones antes de instalar o actualizar un paquete**. Para actualizar la lista de los paquetes disponibles desde los repositorios, previamente a la instalación o actualización, deberíamos teclear el siguiente comando:

```
$ sudo apt-get update
```

Este comando actualiza el índice de paquetes. El índice de APT es esencialmente una base de datos de paquetes disponibles desde los repositorios definidos en el archivo */etc/apt/sources.list* y en el directorio */etc/apt/sources.list.d*

El fichero **/etc/apt/sources.list** es el que se encarga de administrar los repositorios de dónde deben ser descargados los paquetes.

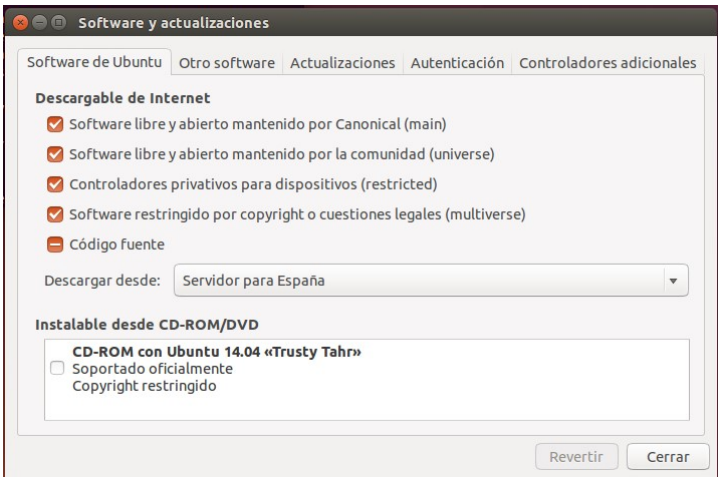
Unas líneas típicas que nos podemos encontrar en este fichero son:

```
deb http://es.archive.ubuntu.com/ubuntu/ xenial main universe
deb-src http://es.archive.ubuntu.com/ubuntu/ xenial main universe
deb http://es.archive.ubuntu.com/ubuntu/ xenial-updates main universe
deb-src http://es.archive.ubuntu.com/ubuntu/ xenial-updates main
      universe
deb cdrom:[Ubuntu 16.04.1 LTS _ Xenial Xerus_ - Release amd64
      (20160719.1)]/ xenial main restricted
```

Donde la información de estas líneas se interpreta:

| | |
|---|---|
| deb | El primer elemento describe el tipo del repositorio. Los valores posibles son **deb** (Paquetes binarios de Debian), **deb-src** (paquetes de fuentes de Debian), **rpm** (paquetes binarios RPM), **rpm-src** (paquetes de fuentes de Redhat), **rpmdir** (carpeta que contiene paquetes binarios RPM) y **rpmdir-src** (carpeta que contiene paquetes de fuentes RPM). |
| http://es.archive.ubuntu.com/ubuntu/ | El segundo elemento es la ubicación del repositorio. Se puede acceder al repositorio por los protocolos HTTP y FTP, o localmente en un CD, DVD o disco duro. |
| xenial | El tercer elemento describe la distribución para la que los paquetes están hechos. Es el nombre en clave de la distribución Ubuntu. |
| main universe | Indica el tipo de repositorio, puede ser **main** (soportado por Canonical en el caso de Ubuntu), **universe** (soportado por la comunidad de usuarios Ubuntu), **restricted** (controladores privativos), **multiverse** (sw restringido por copyright). |

El contenido de este fichero es el que marca la apariencia de la ventana de configuración de la actualización del software que hemos visto anteriormente en este mismo apartado, pero en la pestaña **Software de Ubuntu**.



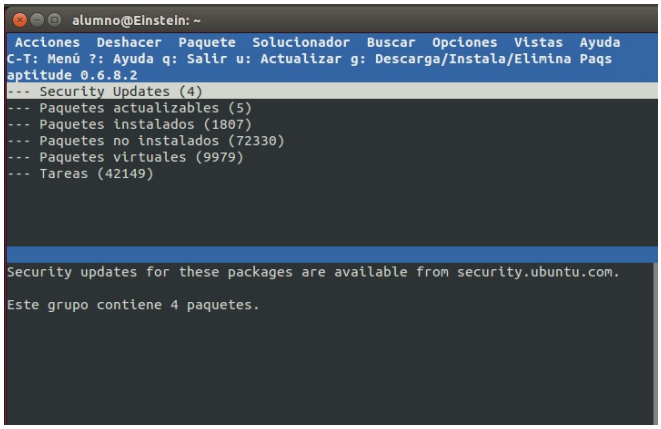

## *aptitude*

*aptitude* es un programa muy similiar a *apt-get* que también se ejecuta en modo terminal. Es muy potente y permite también añadir y quitar aplicaciones del sistema entre otras cosas. a*ptitude* es más completo que *apt-get*, recuerda las librerías

descargadas y las desinstala si están en desuso; en ambas el funcionamiento y la sintaxis es similar. En algunos manuales se recomienda hacerlo con *aptitude* por el mejor manejo de dependencias.

Podemos ejecutar *aptitude* desde la línea de mandatos sin ningún parámetro, apareciéndonos una interfaz de menús.

```
$ aptitude
```




O podemos ejecutar *aptitude* como herramienta de comandos en línea con la sintaxis similar a la de *apt-get*:

```
$ sudo aptitude install nmap
```

## Archivos RPM

Los archivos con extensión *.rpm* son paquetes de aplicaciones preparados para instalarse con el gestor de paquetes de Red Hat. Ubuntu ya hemos visto que no utiliza el gestor de paquetes RPM, sino el *dpkg*, pero en distribuciones basadas en debian (como Ubuntu) se pueden transformar los paquetes *.rpm* en paquetes *.deb* mediante la orden alien:

```
$ sudo alien <paquete>.rpm
```

## Archivos TAR

Son archivos listos para ser desagrupados (y descomprimidos si es necesario) que contienen los ejecutables necesarios para su instalación o ejecución. A veces se encuentran aplicaciones que no proporcionan paquetes de instalación, y hay que compilar a partir del código fuente. En ese caso suelen llevar un fichero con

instrucciones de compilación. Los archivos *.tar* (o *.tar.gz* o *.tar.bz2*) han de descomprimirse con el comando que ya conocemos *tar.*

### Archivos binarios

Los archivos con extensión *.bin* son los programas ejecutables en Linux. 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. Cuando descargamos un archivo de este tipo y lo guardamos en el sistema, no tendrá permiso para ejecutarse. Para ejecutarlos hay que darles permisos de ejecución y después invocarlos.

### Archivos Run

Los archivos con extensión *.run* suelen ser los asistentes para la instalación en Linux. Los archivos *.run* son asistentes, normalmente gráficos, que ayudan a la instalación.

Para ejecutarlos basta con introducir en el terminal:

```
$ sh ./<archivo>.run
```

## 9. Programación de tareas

En Linux existe una utilidad para poder planificar tareas, similar a la existente en Windows, sería el equivalente a las *Tareas Programadas*. Como todo en Linux, esta utilidad es un comando de línea de texto, llamado **cron**. Existen equivalencias en el entorno gráfico, que en el caso de Ubuntu era una utilidad llamada *gnome-schedule*, pero que en la versión 16.04 ha desaparecido de los repositorios.

El programador de tareas *cron* permite ejecutar en segundo plano procesos o scripts a intervalos regulares (por ejemplo, cada minuto, día, semana o mes) de manera repetitiva o en un momento único determinado.

Estos programadores de tareas (el comando cron y sus equivalentes gráficos) permiten ejecutar en segundo plano procesos o scripts a intervalos regulares (por ejemplo, cada minuto, día, semana o mes) de manera repetitiva o en un momento único determinado.

Cron es un demonio (servicio), lo que significa que solo requiere ser iniciado una vez, generalmente con el mismo arranque del sistema. El servicio de cron se llama *crond*. En la mayoría de las distribuciones de Linux el servicio se instala automáticamente y queda iniciado desde el arranque del sistema, se puede comprobar de la siguiente manera:

```
$ sudo /etc/init.d/cron status
```

O bien:

```
$ sudo service cron status
● cron.service - Regular background program processing daemon
   Loaded: loaded (/lib/systemd/system/cron.service; enabled; vendor
preset: enabled)
   Active: active (running) since vie 2017-01-13 15:13:56 CET; 2 weeks
4 days ago
     Docs: man:cron(8)
 Main PID: 1646 (cron)
   CGroup: /system.slice/cron.service
           └─1646 /usr/sbin/cron -f

ene 13 15:13:56 Ubuntu-VirtualBox systemd[1]: Started Regular
background pro....
ene 13 15:13:56 Ubuntu-VirtualBox cron[1646]: (CRON) INFO (Running
@reboot jobs)
Hint: Some lines were ellipsized, use -l to show in full.
```

Los procesos a ejecutar así como la programación de estos se establece en el fichero */etc/crontab*, el cual se puede modificar con un editor de texto (vi, nano, …).

Linux es un sistema multiusuario y cron es de las aplicaciones que soporta el trabajo con varios usuarios a la vez. Cada usuario puede tener su propio archivo *crontab*, de hecho el */etc/crontab* se asume que es el archivo *crontab* del sistema, aunque no hay problema que se incluyan otros usuarios, y de ahí el sexto campo que indica precisamente quién es el usuario que ejecuta la tarea y es obligatorio en */etc/crontab*.

**Nosotros evitaremos trabajar directamente con el fichero *<u>/etc/crontab</u>* del sistema, aunque podamos hacerlo. Cuando los usuarios normales (e incluso *root*) desean generar su propio archivo de crontab, entonces utilizaremos el comando *crontab*:**

```
$ crontab -e
```

En el directorio */var/spool/cron/crontabs* (puede variar según la distribución), se genera un archivo cron para cada usuario, este archivo aunque es de texto, no debe editarse directamente. Así por cada usuario tendríamos un archivo ***/var/spool/cron/crontabs/usuario.*** Este archivo *crontab* se genera directamente con el comando `crontab -e`.

Así se abrirá el editor por defecto (generalmente nano) con el archivo llamado *crontab* vacío y donde el usuario ingresará su tabla de tareas y que se guardará automáticamente como */var/spool/cron/crontabs/usuario*.

Un contenido típico del fichero crontab (en este caso el del sistema):

```
# cat /etc/crontab

SHELL=/bin/bash
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root
HOME=/

# run-parts
01 * * * * root run-parts /etc/cron.hourly
02 4 * * * root run-parts /etc/cron.daily
22 4 * * 0 root run-parts /etc/cron.weekly
42 4 1 * * root run-parts /etc/cron.monthly
```

Las primeras cuatro líneas son variables que indican lo siguiente:

**SHELL** es el 'shell' bajo el cual se ejecuta el cron. Si no se especifica, se tomará por defecto el indicado en la línea */etc/passwd* correspondiente al usuario que este ejecutando cron.

**PATH** contiene o indica la ruta a los directorios en los cuales cron buscará el comando a ejecutar. Este path es distinto al path global del sistema o del usuario.

**MAIL TO** es a quien se le envía la salida del comando (si es que este tiene alguna salida). Cron enviará un correo a quien se especifique en este variable, es decir, debe ser un usuario válido del sistema o de algún otro sistema. Si no se especifica, entonces cron enviará el correo al usuario propietario del comando que se ejecuta.
**HOME** es el directorio raíz o principal del comando cron, si no se indica entonces, la raíz será la que se indique en el archivo */etc/passwd* correspondiente al usuario que ejecuta cron.

Los comentarios se indican con # al inicio de la línea.

Después de lo anterior vienen las líneas que ejecutan las tareas programadas propiamente. No hay límites de cuantas tareas pueda haber, una por renglón. Los campos (son 7) que forman estas líneas están formados de la siguiente manera:

Minuto Hora DiaMes Mes DiaSemana Usuario Comando Campo Descripción

- **Minuto** Controla el minuto de la hora en que el comando será ejecutado, este valor debe de estar entre 0 y 59.
- **Hora** Controla la hora en que el comando será ejecutado, se especifica en un formato de 24 horas, los valores deben estar entre 0 y 23, 0 es medianoche.
- **Día del Mes** Día del mes en que se quiere ejecutar el comando. Por ejemplo se indicaría 20, para ejecutar el comando el día 20 del mes.
- **Mes** Mes en que el comando se ejecutará, puede ser indicado numéricamente (1-12), o por el nombre del mes en inglés, solo las tres primeras letras.
- **Día de la semana** Día en la semana en que se ejecutará el comando, puede ser numérico (0-7) o por el nombre del día en inglés, solo las tres primeras letras. (0 y 7 = domingo)
- **Usuario** Usuario que ejecuta el comando.
- **Comando** Comando, script o programa que se desea ejecutar. Este campo puede contener múltiples palabras y espacios.

Un asterisco * como valor en los primeros cinco campos, indicará inicio-fin del campo, es decir, todo. Un * en el campo de minuto indicará todos los minutos.

Para entender bien esto de los primeros 5 campos y el asterisco vemos varios ejemplos:

| Ejemplo | Descripción |
|---|---|
| 01 * * * * | Se ejecuta al minuto 1 de cada hora de todos los días |
| 15 8 * * * | A las 8:15 a.m. de cada día |
| 15 20 * * * | A las 8:15 p.m. de cada día |
| 00 5 * * 0 | A las 5 a.m. todos los domingos |
| * 5 * * Sun | Cada minuto de 5:00a.m. a 5:59a.m. todos los domingos |
| 45 19 1 * * | A las 7:45 p.m. del primero de cada mes |
| 01 * 20 7 * | Al minuto 1 de cada hora del 20 de julio |
| 10 1 * 12 1 | A la 1:10 a.m. todos los lunes de diciembre |
| 00 12 16 * Wen | Al mediodía de los días 16 de cada mes y que sea Miércoles |
| 30 9 20 7 4 | A las 9:30 a.m. del dia 20 de julio y que sea jueves |
| 30 9 20 7 * | A las 9:30 a.m. del dia 20 de julio sin importar el día de la semana |
| 20 * * * 6 | Al minuto 20 de cada hora de los sábados |
| 20 * * 1 6 | Al minuto 20 de cada hora de los sábados de enero |

También es posible especificar listas en los campos. Las listas pueden estar en la forma de 1,2,3,4 o en la forma de 1-4 que sería lo mismo. Cron, de igual manera soporta incrementos en las listas, que se indican de la siguiente manera:
Valor o lista/incremento

De nuevo, es más fácil entender las listas e incrementos con ejemplos:

| Ejemplo | Descripción |
|---|---|
| 59 11 * 1-3 1,2,3,4,5 | A las 11:59 a.m. de lunes a viernes, de enero a marzo |
| 45 * 10-25 * 6-7 | Al minuto 45 de todas las horas de los días 10 al 25 de todos los meses y que el día sea sábado o domingo |
| 10,30,50 * * * 1,3,5 | En el minuto 10, 30 y 50 de todas las horas de los días lunes, miércoles y viernes |
| */15 10-14 * * * | Cada quince minutos de las 10:00a.m. a las 2:00p.m. |
| * 12 1-10/2 2,8 * | Todos los minutos de las 12 del día, en los días 1,3,5,7 y 9 de febrero y agosto. (El incremento en el tercer campo es |

| | |
|---|---|
| | de 2 y comienza a partir del 1) |
| 0 */5 1-10,15,20-23 * 3 | Cada 5 horas de los días 1 al 10, el día 15 y del día 20 al 23 de cada mes y que el día sea miércoles |
| 3/3 2/4 2 2 2 | Cada 3 minutos empezando por el minuto 3 (3,6,9, etc.) de las horas 2,6,10, etc (cada 4 horas empezando en la hora 2) del día 2 de febrero y que sea martes |

Como se puede apreciar en el último ejemplo la tarea cron que estuviera asignada a ese renglón con esos datos, solo se ejecutaría si se cumple con los 5 campos (AND). Es decir, para que la tarea se ejecute tiene que ser un martes 2 de febrero a las 02:03. Siempre es un AND booleano que solo resulta verdadero si los 5 campos son ciertos en el minuto específico.

Ejemplos de comandos:

```
24 12 * * 1 /usr/bin/who >> /home/conectados.txt
```

Ejecuta la orden who todos los lunes a las 12:24 y guarda la salida en el archivo conectados.txt

```
30 21 * * 6 /sbin/shutdown -h now
```

Todos los sábados a las 21,30 apaga la máquina.

Se pueden ejecutar 2 comandos o más separados por puntos y coma (;):

```
30 15 * * * cd /home/ciclom/Descargas ;wget http://www.asi.com/doc.pdf
```

Descarga el archivo doc.pdf del servidor www.asi.com todos los días a las 15:30

Además de este método, cada hora, día, semana o mes se ejecuta de manera automática lo que hay en los siguientes directorios:

- /etc/cron.hourly
- /etc/cron.daily
- /etc/cron.weekly
- /etc/cron.monthly

Así, bastaría con dejar un script en uno de esos directorios para que se ejecute de manera automática cada *hora, día, semana o mes*, dependiendo del directorio en el que lo hayamos dejado.

Por último, decir que cron permite controlar qué usuarios pueden o no pueden usar los servicios sus servicios. Esto se logra de una manera muy sencilla a través de los siguientes archivos:

- /etc/cron.allow
- /etc/cron.deny

Para impedir que un usuario utilice cron, o mejor dicho el comando crontab, basta con agregar su nombre de usuario al archivo */etc/cron.deny*. Para permitirle su uso entonces sería agregar su nombre de usuario en */etc/cron.allow*. Si por alguna razón se desea negar el uso de cron a todos los usuarios, entonces se puede escribir la palabra ALL al inicio de cron.deny y con eso bastaría.

```
# echo ALL >>/etc/cron.deny
```

o para agregar un usuario más a cron.allow

```
# echo juan >>/etc/cron.allow
```

Si no existe el archivo *cron.allow* ni el archivo *cron.deny*, en teoría el uso de cron está entonces sin restricciones de usuario. Si se añaden nombres de usuarios en *cron.allow*, sin crear un archivo *cron.deny*, tendrá el mismo efecto que haberlo creado con la palabra *ALL*. Esto quiere decir que una vez creado *cron.allow* con un solo usuario, siempre se tendrán que especificar los demás usuarios que se quiere usen cron, en este archivo.

# 10. Demonios

En Linux, se conocen como demonios (Daemons en inglés, no Demons que sería la traducción) a ciertos programas que se están ejecutando en segundo plano (en background), residentes en memoria que interactúan con el kernel y no a nivel usuario y que normalmente proporcionan servicios al usuario (cron, servicios de Internet, …).

La palabra daemon viene de sus siglas en inglés Disk And Execution MONitor.

Los demonios se ejecutan solos. Se inician al arrancar el sistema, si bien el usuario puede arrancarlos y pararlos cuando desee.

Los servicios (demonios) que están disponibles se encuentran en el directorio **/etc/init.d**

Dependiendo de la distribución de Linux, la versión de ésta y del servicio, existen 2 métodos para iniciar o parar servicios, el tradicional o *SysV* (*/etc/init.d*) y el *Upstart*.

Para iniciar un servicio ejecutamos:

```
Tradicional (SysV):
# /etc/init.d/nombre_servicio start
Upstart:
#service nombre_servicio start
```

Para detener un servicio ejecutamos:

```
Tradicional (SysV):
# /etc/init.d/nombre_servicio stop
Upstart:
#service nombre_servicio stop
```

```
$ sudo /etc/init.d/ssh start        # Arrancaría el servidor ssh en
                                     versión tradicional
$ sudo service ssh start            # Arrancaría el servidor ssh en
                                     versión upstart

$ sudo /etc/init.d/webmin stop      # Pararía el servicio webmin
```

Para sacar una lista de los servicios instalados:

```
Tradicional (SysV):
$ sudo ls /etc/init.d
Upstart:
$ sudo service --status-all   #Muestra los servicios de ambos modos
```

Para determinar si un servicio se encuentra o no ejecutándose en un momento dado se debe realizar el siguiente procedimiento:

Si el servicio se encuentra basado en *SysV*.

```
$ sudo /etc/init.d/nombre_servicio status
```

Si el servicio se encuentra basado en *Upstart* se debe hacer lo siguiente.

```
$ sudo status nombre_servicio
```

Vamos a ver cómo hacer que los servicios arranquen de forma automática al iniciar el sistema. Existen también dos métodos, dependiendo de si el servicio está basado en *SysV* o en *Upstart*.

Cuando Linux arranca, puede hacerlo de 7 modos distintos, numerados del 0 al 6. A estos modos se les denomina niveles de ejecución (runlevel) y son los siguientes (dependiendo de la distribución):

- Nivel 0 (Halt): Detiene el sistema. Es un modo de ejecución transitorio.
- Nivel 1 (Monousuario): Permite entrar en el sistema como root sin contraseña y en modo texto. Dependiendo de la distribución se inician unos servicios u otros.
- Nivel 2: En la mayoría de distribuciones se deja sin definir. No en Debian y sus derivados como Ubuntu.
- Nivel 3 (Multiusuario texto): En la mayoría de distribuciones se trata de un modo multiusuario con consola de terminal texto.
- Nivel 4: En la mayoría de distribuciones se deja sin definir. No en Debian y sus derivados como Ubuntu.
- Nivel 5: Mismo funcionamiento que Nivel 3 pero con entorno gráfico. Así arrancan los PCs de usuario.
- Nivel 6 (Reboot): Reinicia el sistema.
- Nivel 2 al 5 (Multiusuario con red y modo gráfico): En Debian y derivados (Ubuntu) arranca en multiusuario en entorno gráfico si está instalado. Se comporta igual en todos esos niveles. A diferencia del resto de distribuciones, no diferencia entre el 3 y el 5.

Por defecto, Ubuntu arranca en el runlevel 2. Para visualizar el runlevel que está arrancado ejecutamos:

```
$ runlevel
```

O bien:

```
$ who -r
```

En función del nivel de ejecución, existe la posibilidad de configurar qué servicios deben iniciarse de forma automática.

Si el servicio está basado en */etc/init.d* (SysV) es necesario crear unos enlaces simbólicos en las carpetas */etc/rcN.d* (donde N es un número de 0 a 6 que indica el nivel de ejecución de linux) que apunten al script de inicio del servicio que se encuentra en */etc/init.d/*. Dichos enlaces deberán tener un nombre un poco especial ya que deberán comenzar con la letra 'S' de Start (arrancar) seguida de un número de dos cifras (para establecer el orden de arranque de los servicios) y del nombre del servicio. Ejemplo: S20samba ó S30nfs. Si lo que nos interesa es que el servicio no arranque, la primera letra deberá ser una K de Kill (detener) en lugar de una S, ejemplo: K20samba ó K30nfs.

Estos enlaces se pueden crear con el comando *update-rc.d*. Ejemplo, si deseamos que el servicio samba se arranque cuando el servidor inicia en los niveles 3, 4 y 5 y no arranque cuando inicia en los niveles 1, 2 y 6, ejecutaremos el siguiente comando (Ojo, no olvidar el punto del final (.) al escribir el comando):

Crear enlaces para inicio automático del servicio samba:

```
$ sudo update-rc.d samba start 20 3 4 5 . stop 20 1 2 6 .
```

De esta forma se crearán enlaces simbólicos de arranque con nombre *S20samba* en las carpetas */etc/rc3.d, /etc/rc4.d y /etc/rc5.d* y de parada con nombre *K20samba* en las carpetas */etc/rc1.d,/etc/rc2.d y /etc/rc6.d*.

El número 20 indica la prioridad. Sirve para arrancar o parar antes unos servicios que otros ya que los scripts se procesan por orden alfabético. Se puede utilizar cualquier número entre 10 y 99.
Si por alguna razón el comando *update-rc.d* no crea los enlaces porque ya están creados, existe la posibilidad de eliminarlos con la opción '-f' (forzado) y acto seguido volver a crearlos:

// Eliminación forzosa de enlaces para inicio automático del servicio samba:

```
# update-rc.d -f samba remove
```

// Volver a crear enlaces para inicio automático del servicio samba

```
# update-rc.d samba start 20 3 4 5 . stop 20 1 2 6 .
```

O de modo más sencillo:

```
$ sudo update-rc.d samba defaults   #en el rl por defecto
$ sudo update-rc.d samba enable
```

Para hacer que no arranque un servicio, otra opción es situarnos en el directorio *rcN.d* que queremos modificar y cambiar el nombre a los enlaces mediante el mandato *mv*.

// No arrancar para el runlevel 5 el servicio samba

```
# cd /etc/rc5.d
# mv S20samba K20samba
```

Si el servicio está en versión *Upstart*, para activar o desactivar los servicios basados en sistema se debe editar el archivo */etc/init/nombre_servicio.conf* y comentar la línea que empieza con *start on* o *stop on*, dependiendo de lo que queramos. Por ejemplo, para desactivar el servicio *cron* se debe realizar el siguiente procedimiento.

```
$ sudo nano /etc/init/cron.conf
```

```
# cron - regular background program processing daemon
# cron is a standard UNIX program that runs user-specified programs at
# periodic scheduled times
description    "regular background program processing daemon"

# start on runlevel [2345]
stop on runlevel [2345]

expect fork
respawn
exec cron
```

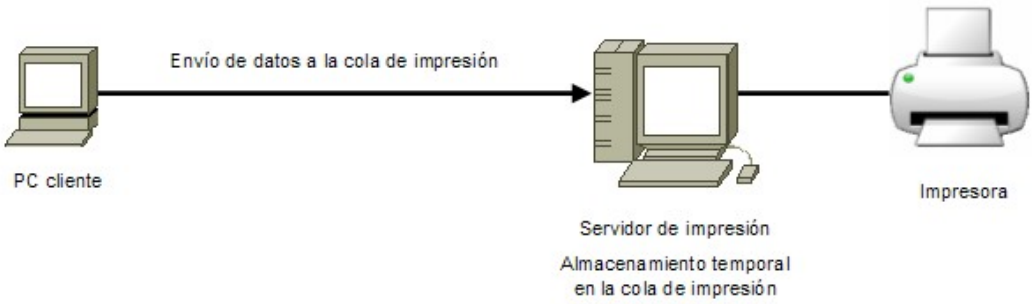
# 11. Gestión de impresión

En un sistema informático es muy frecuente la necesidad de imprimir documentos ya que es una de las aplicaciones principales de los ordenadores. Con la generalización de las redes locales se han sofisticado los sistemas para compartir y optimizar el uso de impresoras. En la actualidad, esos sistemas están muy desarrollados gracias a los servidores de impresión.

Un servidor de impresión es un software que permite que los PCs de una red local puedan hacer uso de las impresoras de la red de una forma eficaz ya que centraliza las tareas de impresión facilitando una gestión de las mismas.

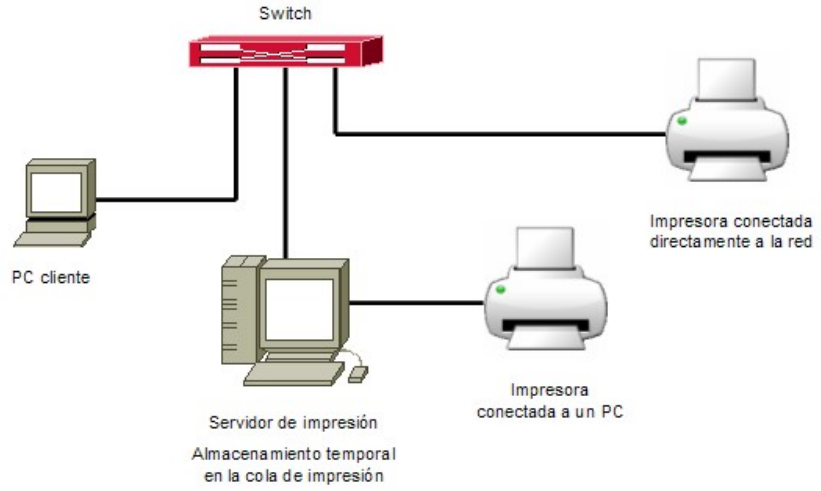
Cuando distintos usuarios desean imprimir documentos, podrían enviarlos directamente hacia la impresora, pero eso consumiría recursos de sus PCs y mezclaría distintos trabajos. Una cola de impresión es un almacén temporal donde permanecen los documentos en espera de que puedan ser imprimidos según un orden secuencial.

La cola de impresión (almacén temporal) puede estar en el propio PC del usuario, en un servidor de impresión o en la misma impresora de red. Lo mejor es que la cola esté en un servidor de impresión, de esa forma el PC del usuario queda menos cargado, los trabajos de impresión de distintos usuarios no se mezclan y existe la posibilidad de administrar los trabajos de impresión (establecer prioridades, límites, controles, etc...)




Las impresoras pueden conectarse a un sistema básicamente de dos formas:

- Impresora conectada a un PC (por puerto paralelo o por USB)
- Impresora conectada directamente a la red




Cuando la impresora está conectada a un equipo, es necesario que dicho equipo esté encendido y que disponga de un software que comparta la impresora para que pueda ser utilizada por el resto de equipos de la red local. Habitualmente las impresoras conectadas a un equipo, suelen estar conectadas a un servidor ya que suelen estar siempre encendidos y además, como hemos comentado anteriormente, lo ideal es que la cola de impresión esté en el servidor.

Las impresoras conectadas directamente a la red son impresoras que disponen de una interfaz ethernet y tienen incorporado el protocolo TCP/IP que les permite integrarse perfectamente en nuestra red local. Suelen disponer de una pequeña pantalla con unos botones para poder configurar la dirección IP. Una vez hayamos configurado la dirección IP, desde un navegador podremos ir a *http://ip-de-la-impresora* para configurar el resto de parámetros y administrarla vía web. Cada vez es más frecuente ver impresoras con servidor de impresión propio aunque si no tienen esa funcionalidad, habrá que configurarla en un servidor de impresión quien administrará la cola de impresión.

**Instalación y configuración de un servidor de impresión.**

Aunque Linux dispone de otros sistemas de impresión, uno muy utilizado es el sistema CUPS (Common Unix Printer System - Sistema de impresión común en Unix). Este sistema de impresión es una capa de impresión libre y portable, y se ha convertido en el estándar para impresión en la mayoría de las distribuciones de Linux.

CUPS gestiona los trabajos y tareas de impresión, y proporciona impresión de red utilizando el Protocolo estándar de Impresión en Internet (IPP), que dispone de soporte para una gran gama de impresoras, desde matriciales hasta láser. CUPS también soporta PostScript Printer Description (PPD) y autodetección de impresoras de red, y dispone de una sencilla herramienta basada en web para la configuración y administración.

El software CUPS permite instalar, configurar, administrar y compartir impresoras en un servidor Linux de una forma bastante sencilla. Este software podrá satisfacer plenamente las necesidades de servidor de impresión que se puedan dar en un sistema informático mediano.

Normalmente el servidor CUPS viene instalado y arrancado. Para instalarlo lo hacemos mediante apt-get el paquete cups:

// Instalación del servidor cups

```
# apt-get install cups
```

El servidor cups, al igual que todos los servicios en Debian, dispone de un script de arranque y parada en la carpeta **/etc/init.d**.

// Iniciar o Reiniciar el servidor cups:

```
#  service cups restart
```

// Parar el servidor cups:

```
# service cups stop
```

Todos los archivos de configuración de cups se encuentran en la carpeta */etc/cups*. El archivo de configuración del servicio es el archivo */etc/cups/cupsd.conf* pero apenas es necesario cambiar nada ya que la configuración del servicio se realiza via web.

Una vez que tenemos en marcha el servicio de impresión cups, podremos configurar impresoras y administrar tareas de impresión. Desde el servidor, debemos abrir un navegador e ir a la siguiente dirección:

http://localhost:631/

Una vez que ya tenemos una impresora configurada en el servidor de impresión, ya estamos en disposición de utilizarla tanto desde el propio servidor como desde el resto de los equipos de la red. Tan solo falta configurarla en los PCs clientes para poder utilizarla.

Para poder utilizar el sistema cups en el resto de PCs de nuestra red, es necesario instalar y configurar el cliente cups. Para instalar el cliente de impresión cups debemos instalar (si no lo está ya) mediante apt-get el paquete cupsys-client que contiene el software necesario para poder imprimir a través de un servidor de impresión cups.

// Instalación del cliente cupsys

```
# apt-get install cupsys-client
```

El archivo de configuración del cliente cups es el archivo **/etc/cups/client.conf**. Si dicho archivo no existe, debemos crearlo con un editor de texto. En dicho archivo tan solo hay que indicar quién es el servidor cups en el parámetro *ServerName*.

// Configuración del cliente cups. Crear archivo */etc/cups/client.conf*

```
ServerName 192.168.1.239
```

De ésta manera, todos los comandos de impresión funcionarán en nuestro sistema de la misma forma que lo hace en el propio servidor.

**Ejemplos de comandos de administración:**

La administración del servidor de impresión comprende las acciones relacionadas con la configuración de impresoras y gestión de usuarios y permisos para utilizar dichas impresoras. Para realizar la tarea de administración disponemos del comando *lpadmin* que permite crear y eliminar impresoras (aunque es más sencillo hacerlo con la herramienta web) y establecer permisos a usuarios entre otras funciones.

Para permitir el uso de la impresora al usuario *jessica* y al grupo *profesores*:

```
# lpadmin -p Laser1010 -u allow:jessica,@profesores
```

Para establecer límite de uso (en 5 páginas)

```
# lpadmin -p Laser1010 -o job-page-limit=5
```

// Comprobar el estado del servidor de impresión

```
# lpstat -t
el planificador de tareas se está ejecutando
no hay un destino predeterminado del sistema
tipo de conexión para Laser1010: usb://HP/LaserJet%201010
Laser1010 aceptando peticiones desde sáb 01 sep 2015 14:12:01 CEST
la impresora Laser1010 está inactiva.  activada desde sáb 01 sep 2015
14:12:01 CEST
```

// Mostrar todos los dispositivos del servidor de impresión

```
# lpinfo -v
network socket
network beh
direct usb://HP/LaserJet%201010
network http
network ipp
network lpd
direct parallel:/dev/lp0
network smb
```

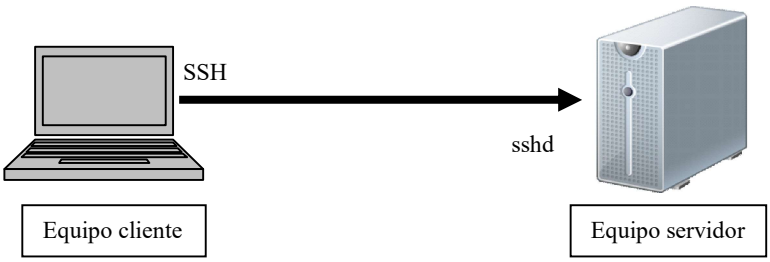
# 12. Administración remota

Ya vimos en el momento de estudiar Windows Server la necesidad de poder administrar remotamente los servidores. Vamos a ver ahora un robusto conjunto de herramientas para el control remoto de equipos conectados en red y la transferencia de datos entre equipos en red, llamado *OpenSSH*.

*OpenSSH* es una versión libre del protocolo Secure Shell (*SSH*) que es una familia de herramientas para control remoto o transferencia de archivos entre equipos. Las herramientas utilizadas tradicionalmente para realizar estas funciones, eran el *telnet* o el *rcp*, que son inseguras y transmiten la contraseña de los usuarios en texto plano cuando son usadas. *OpenSSH* proporciona un demonio y unos clientes para facilitar un control remoto seguro y cifrado, así como operaciones de transferencia de archivos.

El componente servidor de *OpenSSH*, *sshd*, escucha continuamente a la espera de conexiones de clientes desde cualquiera de las herramientas cliente. Cuando aparece una petición de conexión, *sshd* establece la conexión correcta dependiendo del tipo de

herramienta cliente que está conectándose. Por ejemplo, si el equipo remoto se está conectando con la aplicación cliente *ssh*, el servidor *OpenSSH* establecerá una sesión de control remoto tras la autenticación. Si el usuario remoto se conecta al servidor *OpenSSH* con *scp*, el demonio del servidor *OpenSSH* iniciará una copia segura de archivos entre el servidor y el cliente tras la autenticación. *OpenSSH* puede usar muchos métodos de autenticación, incluyendo contraseñas planas, claves públicas y tickets de *Kerberos*




En Ubuntu vienen instalados por defecto tanto el cliente como el servidor OpenSSH. En caso de que no lo estuvieran, la instalación de cliente y servidor es simple. Para instalar las aplicaciones cliente de OpenSSH :

```
$ sudo apt-get install openssh-client
```

Para instalar la aplicación servidor de OpenSSH y los archivos de soporte relacionados:

```
$ sudo apt-get install openssh-server
```

Podemos configurar el comportamiento predeterminado del servidor OpenSSH, sshd, editando el archivo */etc/ssh/sshd_config*. Una vez hechos los cambios habría que reiniciar el servidor sshd:

```
$ sudo service ssh restart
```

Para iniciar una sesión remota desde un equipo cliente hay que indicar la dirección IP (o nombre del equipo), el usuario y la contraseña, además de otros parámetros. Si no se indica usuario y contraseña se intentará conectar con las mismas credenciales que estamos utilizando. En caso de que no coincidan, nos las solicitará:

```
$ ssh 192.168.200.190          #Intenta abrir sesión utilizando usuario
                  y contraseña de la sesión abierta en el cliente

$ ssh juan@192.168.200.190     #Intenta abrir sesión con el usuario juan
```

Podemos incluso, si nos conectamos desde una terminal gráfica, iniciar programas que requieran de una interfaz gráfica utilizando el parámetro –X al conectarnos:

```
$ ssh 192.168.200.190 -X
```

Material elaborado a partir de:

- LPIC-1 Guía de estudio. Christine Bresnahan, Richard Blum. Anaya Multimedia.
- Manual de Ubuntu: Primeros Pasos con Ubuntu 13.10
- Guía de Ubuntu Server https://help.ubuntu.com
- Apuntes de Sistemas Informáticos. DAW. José Ramón Pellicer. IES Camp de Morvedre.
- Administració de la Informació. Implantació de Sistemes Operatius. José Luis Antúnez Reales, Institut Obert Catalunya.
- Apuntes Sistemas Informáticos Monopuesto y Multipuesto. IES Romero Vargas.
- Apuntes Implantación de Sistemas Operativos. José Antonio Carrasco Díaz. IES Romero Vargas.
- Elaboración propia.